LargeDumpling’s Template for Programing Contest

# Data Structure

## Segment Tree

### Classical Segment Tree

1. **const** **int** MAXN=200050;
2. **int** n,m,L[MAXN<<2],R[MAXN<<2];
3. **long** **long** num[MAXN],d[MAXN<<2],tag[MAXN<<2];
4. #define lch ((root)<<1)
5. #define rch ((root)<<1|1)
6. **void** maintain(**int** root)
7. {
8. **if**(L[root]==R[root]) **return**;
9. d[root]=d[lch]+d[rch];
10. **return**;
11. }
12. **void** down(**int** root)
13. {
14. **if**(!tag[root]) **return**;
15. **if**(L[root]==R[root])
16. {
17. tag[root]=0;
18. **return**;
19. }
20. d[lch]+=tag[root]\*(R[lch]-L[lch]+1LL);
21. d[rch]+=tag[root]\*(R[rch]-L[rch]+1LL);
22. tag[lch]+=tag[root];
23. tag[rch]+=tag[root];
24. tag[root]=0;
25. **return**;
26. }
27. **void** build(**int** root,**int** l,**int** r)
28. {
29. L[root]=l; R[root]=r;
30. tag[root]=0;
31. **if**(l==r)
32. {
33. d[root]=num[l];
34. **return**;
35. }
36. **int** mid=(l+r)>>1;
37. build(lch,l,mid);
38. build(rch,mid+1,r);
39. maintain(root);
40. **return**;
41. }
42. **void** change(**int** root,**int** l,**int** r,**long** **long** x)
43. {
44. **if**(l<=L[root]&&R[root]<=r)
45. {
46. tag[root]+=x;
47. d[root]+=(R[root]-L[root]+1LL)\*x;
48. **return**;
49. }
50. down(root);
51. **int** mid=(L[root]+R[root])>>1;
52. **if**(l<=mid) change(lch,l,r,x);
53. **if**(mid<r) change(rch,l,r,x); // \*\*\* NO "else" before "if"
54. maintain(root);
55. **return**;
56. }
57. **long** **long** query(**int** root,**int** l,**int** r)
58. {
59. **if**(l<=L[root]&&R[root]<=r) **return** d[root];
60. down(root);
61. **int** mid=(L[root]+R[root])>>1;
62. **long** **long** sum=0;
63. **if**(l<=mid) sum+=query(lch,l,r);
64. **if**(mid<r) sum+=query(rch,l,r); // \*\*\* NO "else" before "if" AGAIN
65. **return** sum;
66. }
67. #undef lch
68. #undef rch

### ZKW Segment Tree

1. /\*
2. \* 1, M should be a power to two that is bigger than n;
3. \* 2, Operations are valid only in the interval [1,M-2], otherwise unexcepted errors may happen.
4. \*/
5. **const** **int** M=131072;
6. **int** data[M<<1];
7. **void** change(**int** pos,**int** x)
8. {
9. data[pos=pos+M]+=x;
10. **for**(pos>>=1;pos;pos>>=1)
11. data[pos]=data[pos<<1]+data[pos<<1|1];
12. **return**;
13. }
14. **int** query(**int** L,**int** R)
15. {
16. **int** sum=0;
17. **for**(L=L+M-1,R=R+M+1;L<R-1;L>>=1,R>>=1)
18. {
19. **if**(!(L&1)) sum+=data[L^1];
20. **if**(R&1) sum+=data[R^1];
21. }
22. **return** sum;
23. }

### Persistent Segment Tree

1. **const** **int** MAXN=100050;
2. **int** d[MAXN\*30],ch[MAXN\*30][2],root[MAXN],sz=0;
3. **void** add(**int** &now,**int** pre,**int** L,**int** R,**int** p,**int** x)
4. {
5. now=++sz;
6. memcpy(ch[now],ch[pre],**sizeof**(ch[pre]));
7. d[now]=d[pre]+x;
8. **if**(L==R) **return**;
9. **int** mid=(L+R)>>1;
10. **if**(p<=mid) add(ch[now][0],ch[pre][0],L,mid,p,x);
11. **else** **if**(mid<p) add(ch[now][1],ch[pre][1],mid+1,R,p,x);
12. **return**;
13. }
14. **int** query(**int** pre,**int** now,**int** L,**int** R,**int** k)
15. {
16. **if**(L==R) **return** L;
17. **int** mid=(L+R)>>1;
18. **if**(k<=d[ch[now][0]]-d[ch[pre][0]]) **return** query(ch[pre][0],ch[now][0],L,mid,k);
19. k-=(d[ch[now][0]]-d[ch[pre][0]]);
20. **return** query(ch[pre][1],ch[now][1],mid+1,R,k);
21. }

## Binary Indexed Tree

1. #define low(x) ((x)&(-x))
2. **const** **int** MAXN=100050;
3. **int** d[MAXN];
4. **void** add(**int** p,**int** x)
5. {
6. **for**(;p<MAXN;p+=low(p))
7. d[p]+=x;
8. **return**;
9. }
10. **int** query(**int** x)
11. {
12. **int** sum=0;
13. **for**(;x;x-=low(x))
14. sum+=d[x];
15. **return** sum;
16. }

## Binary Search Tree

### Treap

1. **struct** jp \*null;
2. **struct** jp
3. {
4. **int** key,num,size,v;
5. jp \*son[2];
6. jp(**const** **int** &X=0):key(X) { num=size=1; v=rand(); son[0]=son[1]=null; }
7. **int** cmp(**const** **int** &x) { **return** x==key?-1:(x<key?0:1); }
8. **void** updata() { size=son[0]->size+son[1]->size+num; }
9. }\*root;
10. **void** adjust(jp\* &r,**int** d)
11. {
12. jp \*u=r->son[d]; r->son[d]=u->son[d^1]; u->son[d^1]=r;
13. r->updata(); (r=u)->updata(); **return**;
14. }
15. **void** insert(jp\* &r,**int** x)
16. {
17. **if**(r==null)
18. {
19. r=**new** jp(x);
20. **return**;
21. }
22. **int** d=r->cmp(x);
23. **if**(d==-1) r->num++;
24. **else**
25. {
26. insert(r->son[d],x);
27. **if**(r->son[d]->v > r->v) adjust(r,d);
28. }
29. r->updata();
30. **return**;
31. }
32. **bool** del(jp\* &r,**int** x)
33. {
34. **if**(r==null) **return** **false**;
35. **int** d=r->cmp(x);
36. **if**(d==-1)
37. {
38. **if**(r->num>1) r->num--;
39. **else**
40. {
41. **if**(r->son[0]==null||r->son[1]==null)
42. {
43. jp \*u=r;
44. r=r->son[0]==null?r->son[1]:r->son[0];
45. **delete** u;
46. }
47. **else**
48. {
49. d=r->son[0]->v < r->son[1]->v;
50. adjust(r,d);
51. del(r->son[d^1],x);
52. }
53. }
54. r->updata();
55. **return** **true**;
56. }
57. **bool** flag=del(r->son[d],x);
58. **if**(flag) r->updata();
59. **return** flag;
60. }
61. **int** rank(jp\* &r,**int** x)
62. {
63. **if**(r==null) **return** 0;
64. **int** d=r->cmp(x);
65. **if**(!d) **return** rank(r->son[0],x);
66. **return** r->son[0]->size+(d==-1?1:rank(r->son[1],x)+r->num);
67. }
68. **int** kth(jp\* &r,**int** rk)
69. {
70. **if**(r==null) **return** 0;
71. **if**(rk<=r->son[0]->size) **return** kth(r->son[0],rk);
72. rk-=(r->son[0]->size+r->num);
73. **return** rk>0?kth(r->son[1],rk):r->key;
74. }
75. **int** query(**int** x,**int** D)
76. {
77. jp \*u=root; **int** ans=0,d;
78. **while**(u!=null)
79. {
80. d=u->cmp(x);
81. **if**(d==(D^1)) ans=u->key;
82. u=u->son[d==-1?D:d];
83. }
84. **return** ans;
85. }
86. root=null=**new** jp();
87. null->son[0]=null->son[1]=null;
88. null->num=null->size=0;

### Splay

1. **struct** jp \*null;
2. **struct** jp
3. {
4. **int** key,size;
5. **bool** flag;
6. jp \*son[2];
7. jp(**const** **int** &x=0):key(x) { size=1; flag=**false**; son[0]=son[1]=null; }
8. **int** cmp(**int** k) { **return** k==(son[0]->size+1)?-1:(k<=son[0]->size?0:1); }
9. **void** updata() { size=son[0]->size+son[1]->size+1; **return**; }
10. **void** down()
11. {
12. **if**(!flag) **return**;
13. flag=**false**;
14. swap(son[0],son[1]);
15. son[0]->flag^=**true**;
16. son[1]->flag^=**true**;
17. **return**;
18. }
19. }\*root;
20. **int** cnt=0,n,m;
21. **void** adjust(jp\* &r,**int** d)
22. {
23. jp\* u=r->son[d]; r->son[d]=u->son[d^1]; u->son[d^1]=r;
24. r->updata(); (r=u)->updata(); **return**;
25. }
26. **void** build(jp\* &u,**int** l,**int** r)
27. {
28. **if**(l>r) { u=null; **return**; }
29. **int** mid=(l+r)>>1;
30. u=**new** jp();
31. build(u->son[0],l,mid-1);
32. u->key=cnt++;
33. build(u->son[1],mid+1,r);
34. u->updata();//勿忘
35. **return**;
36. }
37. **void** m41441(jp\* &r,**int** k)
38. {
39. r->down();
40. **int** d=r->cmp(k);
41. **if**(d==1) k=k-r->son[0]->size-1;
42. **if**(d!=-1)
43. {
44. jp\* u=r->son[d];
45. u->down();
46. **int** d2=u->cmp(k);
47. **int** k2=d2==1?k-u->son[0]->size-1:k;
48. **if**(d2!=-1)
49. {
50. m41441(u->son[d2],k2);
51. **if**(d2==d) adjust(r,d);
52. **else** adjust(r->son[d],d2);
53. }
54. adjust(r,d);
55. }
56. **return**;
57. }
58. **void** split(jp\* all,**int** k,jp\* &left,jp\* &right)
59. {
60. m41441(all,k);
61. left=all;
62. right=all->son[1];
63. left->son[1]=null;
64. left->updata();
65. **return**;
66. }
67. jp \*merge(jp\* left,jp\* right)
68. {
69. m41441(left,left->size);
70. left->son[1]=right;
71. left->updata();
72. **return** left;
73. }
74. **void** rEverse(**int** l,**int** r)
75. {
76. jp \*temp,\*left,\*mid,\*right;
77. split(root,l,left,temp);
78. split(temp,r-l+1,mid,right);
79. mid->flag^=**true**;
80. root=merge(merge(left,mid),right);
81. **return**;
82. }
83. **int** tot=0;
84. **void** print(jp\* &r)
85. {
86. **if**(r==null) **return**;
87. r->down();
88. print(r->son[0]);
89. **if**(r->key)
90. {
91. printf("%d",r->key);
92. **if**(++tot!=n) putchar(' ');
93. }
94. print(r->son[1]);
95. **return**;
96. }
97. root=null=**new** jp();
98. null->son[0]=null->son[1]=null;
99. null->size=0;
100. build(root,0,n);

## Mergeable Heap

### Skew Heap

1. **template**<**typename** Ty>
2. **struct** heap
3. {
4. **static** **const** **int** MAXN=1000050;
5. **int** tOp;
6. **int** L[MAXN],R[MAXN],dist[MAXN],p[MAXN];
7. Ty data[MAXN];
8. heap()
9. {
10. tOp=0;
11. memset(L,0,**sizeof**(L));
12. memset(R,0,**sizeof**(R));
13. memset(dist,0,**sizeof**(dist));
14. **for**(**int** i=1;i<MAXN;i++) p[i]=i;
15. p[0]=MAXN-1;
16. }
17. **void** clear()
18. {
19. tOp=0;
20. memset(L,0,**sizeof**(L));
21. memset(R,0,**sizeof**(R));
22. memset(dist,0,**sizeof**(dist));
23. **for**(**int** i=1;i<MAXN;i++) p[i]=i;
24. p[0]=MAXN-1;
25. **return**;
26. }
27. **bool** empty() { **return** (**bool**)tOp; }
28. **int** merge(**int** A,**int** B)
29. {
30. **if**(!A) **return** B;
31. **if**(!B) **return** A;
32. **if**(data[A]>data[B]) swap(A,B);
33. R[A]=merge(R[A],B);
34. **if**(dist[L[A]]<dist[R[A]])
35. swap(L[A],R[A]);
36. dist[A]=R[A]?dist[R[A]]+1:0;
37. **return** A;
38. }
39. **void** push(Ty X)
40. {
41. data[p[p[0]]]=X;
42. tOp=merge(tOp,p[p[0]--]);
43. **return**;
44. }
45. **void** pop()
46. {
47. p[++p[0]]=tOp;
48. tOp=merge(L[tOp],R[tOp]);
49. L[p[p[0]]]=R[p[p[0]]]=0;
50. **return**;
51. }
52. Ty top() { **return** data[tOp]; }
53. };

# String Algorithm

## The Knuth-Morris-Pratt Algorithm

### Classical KMP

1. **const** **int** MAXL=100050;
2. **char** tex[MAXL],T[MAXL];
3. **int** pre[MAXL];
4. **int** KMP()
5. {
6. **int** n=strlen(tex),m=strlen(T),i,k;
7. pre[0]=-1;
8. **for**(i=1,k=-1;i<n;pre[i++]=k)
9. {
10. **while**(k>=0&&tex[i]!=tex[k+1])
11. k=pre[k];
12. **if**(tex[i]==tex[k+1])
13. k++;
14. }
15. **for**(i=0,k=-1;i<m;i++)
16. {
17. **while**(k>=0&&T[i]!=tex[k+1])
18. k=pre[k];
19. **if**(T[i]==tex[k+1])
20. {
21. k++;
22. **if**(k==n-1) **return** i-k;
23. }
24. }
25. **return** -1;
26. }

### Extended KMP

1. **const** **int** maxn=100010;   //字符串长度最大值
2. **int** next[maxn],ex[maxn]; //ex数组即为extend数组
3. //预处理计算next数组
4. **void** GETNEXT(**char** \*str)
5. {
6. **int** i=0,j,po,len=strlen(str);
7. next[0]=len;//初始化next[0]
8. **while**(str[i]==str[i+1]&&i+1<len)//计算next[1]
9. i++;
10. next[1]=i;
11. po=1;//初始化po的位置
12. **for**(i=2;i<len;i++)
13. {
14. **if**(next[i-po]+i<next[po]+po)//第一种情况，可以直接得到next[i]的值
15. next[i]=next[i-po];
16. **else**//第二种情况，要继续匹配才能得到next[i]的值
17. {
18. j=next[po]+po-i;
19. **if**(j<0)j=0;//如果i>po+next[po],则要从头开始匹配
20. **while**(i+j<len&&str[j]==str[j+i])//计算next[i]
21. j++;
22. next[i]=j;
23. po=i;//更新po的位置
24. }
25. }
26. }
27. //计算extend数组
28. **void** EXKMP(**char** \*s1,**char** \*s2)
29. {
30. **int** i=0,j,po,len=strlen(s1),l2=strlen(s2);
31. GETNEXT(s2);//计算子串的next数组
32. **while**(s1[i]==s2[i]&&i<l2&&i<len)//计算ex[0]
33. i++;
34. ex[0]=i;
35. po=0;//初始化po的位置
36. **for**(i=1;i<len;i++)
37. {
38. **if**(next[i-po]+i<ex[po]+po)//第一种情况，直接可以得到ex[i]的值
39. ex[i]=next[i-po];
40. **else**//第二种情况，要继续匹配才能得到ex[i]的值
41. {
42. j=ex[po]+po-i;
43. **if**(j<0)j=0;//如果i>ex[po]+po则要从头开始匹配
44. **while**(i+j<len&&j<l2&&s1[j+i]==s2[j])//计算ex[i]
45. j++;
46. ex[i]=j;
47. po=i;//更新po的位置
48. }
49. }
50. }

## Manacher

1. **const** **int** MAXL=11000050;
2. **char** str[MAXL<<1];
3. **int** r[MAXL<<1],len;
4. /\*
5. \* 1, str should have double lenth
6. \*/
7. **void** Manacher()
8. {
9. **int** rig=0,id=0;
10. len=strlen(str);
11. str[len<<1]='#';
12. **for**(**int** i=len-1;0<=i;i--)
13. {
14. str[i<<1|1]=str[i];
15. str[i<<1]='#';
16. }
17. r[0]=0;
18. **for**(**int** i=1;i<=(len<<1);i++)
19. {
20. **if**(i<=rig) r[i]=min(rig-i,r[2\*id-i]);
21. **else** r[i]=0;
22. **while**(0<=i-r[i]-1
23. &&i+r[i]+1<=(len<<1)
24. &&str[i-r[i]-1]==str[i+r[i]+1])
25. r[i]++;
26. **if**(rig<i+r[i])
27. {
28. rig=i+r[i];
29. id=i;
30. }
31. }
32. **return**;
33. }
34. **for**(**int** i=0;i<=(len<<1);i++)
35. ans=max(ans,r[i]);

## Aho-Corasick automaton

1. #include<queue>
2. **const** **int** MAXN=100050;
3. **const** **int** MAXC=26;
4. **int** ch[MAXN][MAXC],val[MAXN],last[MAXN],pre[MAXN],sz=0;
5. **void** insert(**char** T[],**int** x)
6. {
7. **int** lenth=strlen(T),u=0;
8. **for**(**int** i=0;i<lenth;u=ch[u][T[i++]-97])
9. **if**(!ch[u][T[i]-97])
10. ch[u][T[i]-97]=++sz;
11. val[u]=x;
12. **return**;
13. }
14. **void** getfail()
15. {
16. **int** u,v,f;
17. queue<**int**>q;
18. **for**(**int** i=0;i<MAXC;i++)
19. {
20. **if**(!ch[0][i])
21. **continue**;
22. pre[ch[0][i]]=last[ch[0][i]]=0;
23. q.push(ch[0][i]);
24. }
25. **while**(q.size())
26. {
27. u=q.front();
28. q.pop();
29. **for**(**int** i=0;i<MAXC;i++)
30. {
31. **if**(!ch[u][i])
32. {
33. ch[u][i]=ch[pre[u]][i];
34. **continue**;
35. }
36. v=ch[u][i];
37. q.push(v);
38. f=pre[u];
39. **while**(f&&!ch[f][i])
40. f=pre[f];
41. pre[v]=ch[f][i];
42. last[v]=val[pre[v]]?pre[v]:last[pre[v]];
43. }
44. }
45. **return**;
46. }
47. **void** count(**int** u)
48. {
49. **if**(u)
50. {
51. // do some operators
52. **if**(last[u]) count(last[u]);
53. }
54. **return**;
55. }
56. **void** find(**char** T[])
57. {
58. **int** lenth=strlen(T),u=0;
59. **for**(**int** i=0;i<lenth;i++)
60. {
61. /\*while(u&&!ch[u][T[i]-97])
62. u=pre[u];\*/
63. u=ch[u][T[i]-97];
64. **if**(val[u]) count(u);
65. **else** **if**(last[u]) count(last[u]);
66. }
67. **return**;
68. }

## Palindrome Automaton

1. **const** **int** MAXN=100050;
2. **const** **int** MAXC=26;
3. **int** ch[MAXN][MAXC],len[MAXN],fail[MAXN],cnt[MAXN],last,sz;
4. **int** getfail(**char** T[],**int** x,**int** i)
5. {
6. **while**(T[i-len[x]-1]!=T[i]) x=fail[x];
7. **return** x;
8. }
9. **void** init()
10. {
11. memset(ch[0],last=0,**sizeof**(ch[0]));//last为当前的最长后缀回文
12. memset(ch[1],0,**sizeof**(ch[1]));
13. len[0]=0; len[1]=-1; fail[0]=1;
14. sz=1;
15. **return**;
16. }
17. **void** insert(**char** T[])
18. {
19. **int** lenth=strlen(T),cur;
20. **for**(**int** i=0;i<lenth;i++)
21. {
22. cur=getfail(T,last,i);
23. **if**(!ch[cur][T[i]-97])
24. {
25. fail[++sz]=ch[getfail(T,fail[cur],i)][T[i]-97];
26. ch[cur][T[i]-97]=sz;
27. memset(ch[sz],0,**sizeof**(ch[sz]));
28. len[sz]=len[cur]+2;
29. }
30. last=ch[cur][T[i]-97];
31. cnt[last]++;
32. }
33. }

## Suffix Array

1. **const** **int** MAXN=100050;
2. **char** str[MAXN];
3. **int** len,sa[MAXN],rank[MAXN],height[MAXN];
4. **int** num[MAXN],bsa[MAXN],c[MAXN];
5. **void** build\_sa()
6. {
7. **int** \*x=num,\*y=bsa,m=26,p,i,k;
8. **for**(i=0;i<m;i++) c[i]=0;
9. **for**(i=0;i<len;i++) c[x[i]=str[i]-'a']++;
10. **for**(i=1;i<m;i++) c[i]+=c[i-1];
11. **for**(i=len-1;0<=i;i--) sa[--c[x[i]]]=i;
12. **for**(k=1;k<=len;k<<=1)
13. {
14. p=0;
15. **for**(i=len-1;len-k<=i;i--) y[p++]=i;
16. **for**(i=0;i<len;i++) **if**(sa[i]>=k) y[p++]=sa[i]-k;
17. **for**(i=0;i<m;i++) c[i]=0;
18. **for**(i=0;i<len;i++) c[x[y[i]]]++;
19. **for**(i=1;i<m;i++) c[i]+=c[i-1];
20. **for**(i=len-1;0<=i;i--) sa[--c[x[y[i]]]]=y[i];
21. swap(x,y);
22. p=1;
23. x[sa[0]]=0;
24. **for**(i=1;i<len;i++)
25. x[sa[i]]=(y[sa[i-1]]==y[sa[i]]&&y[sa[i-1]+k]==y[sa[i]+k])?p-1:p++;
26. **if**(len<=p) **break**;
27. m=p;
28. }
29. **return**;
30. }
31. **void** build\_height()
32. {
33. **int** k=0,j,i;
34. **for**(i=0;i<len;i++) rank[sa[i]]=i;
35. **for**(i=0;i<len;i++)
36. {
37. **if**(k) k--;
38. **if**(!rank[i]) **continue**;
39. j=sa[rank[i]-1];
40. **while**(str[i+k]==str[j+k]) k++;
41. height[rank[i]]=k;
42. }
43. **return**;
44. }

## Minimum representation

1. **int** minP(**char** s[])
2. {
3. **int** l=strlen(s);
4. **int** i=0,j=1,k=0;
5. **while**(**true**)
6. {
7. **if**(i+k>=l||j+k>=l) **break**;
8. **if**(s[i+k]==s[j+k]) k++;
9. **else**
10. {
11. **if**(s[j+k]>s[i+k]) j+=k+1;
12. **else** i+=k+1;
13. k=0;
14. **if**(i==j) j++;
15. }
16. }
17. **return** min(i,j);
18. }

# Mathematics

## Berlekamp-Massey Algorithm

1. #include <cstdio>
2. #include <vector>
3. **using** **namespace** std;
4. **namespace** BerlekampMassey {
5. **const** **int** mod = 1e9 + 7;
6. **int** L, m, b, n;
7. vector<**int**> s, C, B;
8. **void** init() {
9. s.clear();
10. C.clear();
11. B.clear();
12. C.push\_back(1);
13. B.push\_back(1);
14. L = n = 0;
15. m = b = 1;
16. }
17. **int** pow\_mod(**int** a, **int** k) {
18. **int** s = 1;
19. **while** (k) {
20. **if** (k & 1)
21. s = 1ll \* s \* a % mod;
22. a = 1ll \* a \* a % mod;
23. k >>= 1;
24. }
25. **return** s;
26. }
27. **void** update(**int** d) {
28. s.push\_back(d);
29. **for** (**int** i = 1; i <= L; ++i)
30. d = (d + 1ll \* C[i] \* s[n - i] % mod) % mod;
31. **if** (d == 0)
32. ++m;
33. **else** **if** (2 \* L <= n) {
34. vector<**int**> T = C;
35. C.resize(n + 1 - L + 1);
36. **for** (**int** i = L + 1; i <= n + 1 - L; ++i)
37. C[i] = 0;
38. **for** (**int** i = 0; i < B.size(); ++i)
39. C[i + m] = (C[i + m] + mod - 1ll \* d \* pow\_mod(b, mod - 2) % mod \* B[i] % mod) % mod;
40. L = n + 1 - L;
41. B = T;
42. b = d;
43. m = 1;
44. } **else** {
45. **for** (**int** i = 0; i < B.size(); ++i)
46. C[i + m] = (C[i + m] + mod - 1ll \* d \* pow\_mod(b, mod - 2) % mod \* B[i] % mod) % mod;
47. ++m;
48. }
49. ++n;
50. }
51. **void** output() {
52. printf("F(n)=");
53. **for** (**int** i = 1; i < C.size(); ++i) {
54. **int** output = (mod - C[i]) % mod;
55. **if** (output > mod / 2)
56. output -= mod;
57. printf("%s%d\*F(n-%d)", (output < 0 || i == 1) ? "" : "+", output, i);
58. }
59. puts("");
60. }
61. **void** output\_code\_for() {
62. **static** **const** **char** \*name = "dp";
63. **static** **const** **char** \*index = "i";
64. **static** **const** **char** \*upperbound = "maxn";
65. puts("// Generated by Berlekamp-Massey algorithm");
66. **for** (**int** i = 1; i < C.size(); ++i) {
67. printf("%s[%d]=%d;\n", name, i - 1, s[i - 1]);
68. }
69. printf("for(int i=%d;i<%s;++i)\n", (**int**)C.size() - 1, upperbound);
70. printf("  %s[%s]=((", name, index);
71. **for** (**int** i = 1; i < C.size(); ++i) {
72. **int** output = (mod - C[i]) % mod;
73. **if** (output > mod / 2)
74. output -= mod;
75. printf("%s%d\*%s[%s-%d]%%mod", (output < 0 || i == 1) ? "" : "+", output, name, index, i);
76. }
77. puts(")%mod+mod)%mod;");
78. }
79. **void** output\_code\_matrix() {
80. // TODO
81. }
82. };
84. #include "BerlekampMassey.cpp"
85. **int** main() {
86. **int** arr[12] = {2, 24, 96, 416, 1536, 5504, 18944, 64000, 212992, 702464, 2301952, 7512064};
87. BerlekampMassey::init();
88. **for** (**int** i = 0; i < 12; ++i) {
89. BerlekampMassey::update(arr[i]);
90. }
91. printf("Formule: ");
92. BerlekampMassey::output();
93. printf("Code: \n");
94. BerlekampMassey::output\_code\_for();
95. **return** 0;
96. }

## Euclid's algorithm

### Classical

1. /\*
2. \* ax+by=gcd(a,b)=d
3. \*/
4. **void** gcd(**int** a,**int** b,**int** &d,**int** &x,**int** &y)
5. {
6. **if**(!b)
7. {
8. d=a;
9. x=1;
10. y=0;
11. }
12. **else**
13. {
14. gcd(b,a%b,d,y,x);
15. y-=x\*(a/b);
16. }
17. **return**;
18. }

### Quasi Euclid’s algorithm

## Sieve

### Sieve of Eratosthenes

1. **const** **int** MAXN=100050;
2. **bool** vis[MAXN];
3. **void** Eratosthenes()
4. {
5. **int** m=sqrt(n+0.5);
6. memset(vis,**false**,**sizeof**(vis));
7. **for**(**int** i=2;i<=m;i++) **if**(!vis[i])
8. **for**(**int** j=i\*i;j<=n;j+=i) vis[j]=**true**;
9. **return**;
10. }

### Sieve of Euler

1. **const** **int** MAXN=100050;
2. **int** pri[MAXN],phi[MAXN],u[MAXN];
3. **bool** vis[MAXN];
4. **void** Euler()
5. {
6. memset(vis,**true**,**sizeof**(vis));
7. pri[0]=0;
8. u[1]=1; phi[1]=1;
9. **for**(**int** i=2;i<MAXN;i++)
10. {
11. **if**(vis[i])
12. {
13. pri[++pri[0]]=i;
14. phi[i]=i-1;
15. u[i]=-1;
16. }
17. **for**(**int** j=1;j<=pri[0]&&i\*pri[j]<MAXN;j++)
18. {
19. vis[i\*pri[j]]=**false**;
20. **if**(i%pri[j])
21. {
22. phi[i\*pri[j]]=phi[i]\*(pri[j]-1);
23. u[i\*pri[j]]=-u[i];
24. }
25. **else**
26. {
27. phi[i\*pri[j]]=phi[i]\*pri[j];
28. **break**;
29. }
30. }
31. }
32. }

## Möbius inversion

### Divisor

### Multiple

## Gauss-Jordan Elimination

### System of Linear Equations

1. **const** **int** MAXN=1050;
2. **const** **double** eps=1e-8;
3. **typedef** **double** Matrix[MAXN][MAXN];
4. **void** Gauss\_Jordan\_Elimination(Matrix A,**int** n)
5. {
6. **int** i,j,k,r;
7. **for**(i=0;i<n;i++)
8. {
9. r=i;
10. **for**(j=i+1;j<n;j++)
11. {
12. **if**(abs(A[j][i]>fabs(A[r][i]))) r=j;
13. //if(fabs(A[r][i])<eps) continue;
14. **if**(r!=i) **for**(**int** j=0;j<=n;j++)
15. swap(A[r][j],A[i][j]);
16. }
17. **for**(k=0;k<n;k++) **if**(k!=i)
18. **for**(**int** j=n;j>=i;j--)
19. A[k][j]-=A[k][i]/A[i][i]\*A[i][j];
20. }
21. **return**;
22. }

### System of XOR Equations

1. **const** **int** MAXM=512;
2. vector<bitset<MAXM> >equ;
3. **int** Free=0;
4. **void** gauss()
5. {
6. **int** limit=equ.size(),tot=0;
7. **int** j;
8. **for**(**int** i=0;i<limit;i++)
9. {
10. **for**(j=tot;j<limit&&!equ[j][i];j++);
11. **if**(j==limit) Free++;
12. **else**
13. {
14. swap(equ[tot++],equ[j]);
15. **for**(j=tot;j<limit;j++) **if**(equ[j][i])
16. equ[j]^=equ[tot-1];
17. }
18. }
19. **return**;
20. }

## Fast Fourier Transform

1. **struct** cp
2. {
3. **double** r,i;
4. cp(**double** a=0,**double** b=0):r(a),i(b){}
5. cp operator+(cp X)**const**{**return** cp(r+X.r,i+X.i);}
6. cp operator-(cp X)**const**{**return** cp(r-X.r,i-X.i);}
7. cp operator\*(cp X)**const**{**return** cp(r\*X.r-i\*X.i,r\*X.i+i\*X.r);}
8. }A[400050],B[400050];
9. **int** rev[400050];
10. **void** FFT\_pre(**int** &n)
11. {
12. **int** tem; **for**(tem=0;(1<<tem)<n;tem++); n=1<<tem;
13. rev[0]=0;
14. **for**(**int** i=1;i<n;i++) rev[i]=(rev[i>>1]>>1)|((i&1)<<(tem-1));
15. **return**;
16. }
17. **void** DFT(cp X[],**int** n,**bool** inv)
18. {
19. **int** temk;
20. **double** wn1=inv?-2.\*acos(-1.):2.\*acos(-1.);
21. cp w,cur,tem;
22. **for**(**int** i=1;i<n;i++) **if**(rev[i]>i)swap(X[i],X[rev[i]]);
23. **for**(**int** k=2;k<=n;k<<=1)
24. {
25. temk=k>>1;
26. cur=cp(1.,0.);
27. w=cp(cos(wn1/k),sin(wn1/k));
28. **for**(**int** j=0;j<temk;j++)
29. {
30. **for**(**int** i=0;i<n;i+=k)
31. {
32. tem=cur\*X[i+j+temk];
33. X[i+j+temk]=X[i+j]-tem;
34. X[i+j]=X[i+j]+tem;
35. }
36. cur=cur\*w;
37. }
38. }
39. **if**(inv) **for**(**int** i=0;i<n;i++) X[i].r=X[i].r/n;
40. **return**;
41. }
42. **int** main()
43. {
44. **int** n,m,lenth;
45. scanf("%d%d",&n,&m);
46. n++;m++;
47. lenth=n+m-1;
48. **for**(**int** i=0;i<n;i++) scanf("%lf",&A[i].r);
49. **for**(**int** i=0;i<m;i++) scanf("%lf",&B[i].r);
50. n=n<m?m:n;
51. n<<=1;
52. FFT\_pre(n);
53. //for(int i=0;i<n;i++) printf("%d ",rev[i]);
54. DFT(A,n,**false**);
55. DFT(B,n,**false**);
56. **for**(**int** i=0;i<n;i++) A[i]=A[i]\*B[i];
57. DFT(A,n,**true**);
58. **for**(**int** i=0;i<lenth;i++) printf("%d ",(**int**)(A[i].r+0.5));
59. **return** 0;
60. }

## Number Theoretic Transform

1. **const** **int** P=(479<<21)+1,G=3,D=10000000;
2. **int** A[405000],B[405000],rev[400050],n,m,lenth,inver;
3. **int** pOw(**int** a,**int** n)
4. {
5. **int** ans;
6. **for**(ans=1;n;n>>=1,a=(**int**)((**long** **long**)a\*a%P))
7. **if**(n&1)ans=(**int**)((**long** **long**)ans\*a%P);
8. **return** ans;
9. }
10. **void** FFT\_pre(**int** &n)
11. {
12. **int** tem; **for**(tem=0;(1<<tem)<n;tem++); n=1<<tem;
13. **for**(**int** i=0;i<n;i++) rev[i]=rev[i>>1]>>1|((i&1)<<(tem-1));
14. inver=pOw(n,P-2);
15. **return**;
16. }
17. **void** FFT(**int** X[],**int** n,**bool** inv)
18. {
19. **for**(**int** i=0;i<n;i++) **if**(rev[i]>i) swap(X[i],X[rev[i]]);
20. **int** temk,w,cur,temp;
21. **for**(**int** k=2;k<=n;k<<=1)
22. {
23. temk=k>>1;
24. cur=1; w=pOw(G,(P-1)/k);
25. **if**(inv) w=pOw(w,k-1);
26. **for**(**int** i=0;i<temk;i++)
27. {
28. **for**(**int** j=0;j<n;j+=k)
29. {
30. temp=(**int**)((**long** **long**)cur\*X[i+j+temk]%P);
31. X[i+j+temk]=(**int**)((**long** **long**)(X[i+j]-temp+P)%P);
32. X[i+j]=(**int**)((**long** **long**)(X[i+j]+temp)%P);
33. }
34. cur=(**int**)((**long** **long**)cur\*w%P);
35. }
36. }
37. **if**(inv) **for**(**int** i=0;i<n;i++) X[i]=(**int**)((**long** **long**)X[i]\*inver%P);
38. **return**;
39. }
40. **int** main()
41. {
42. read1n(n); read1n(m);
43. n++; m++;
44. lenth=n+m-1;
45. n=max(n,m);
46. n<<=1;
47. FFT\_pre(n);
48. FFT(A,n,**false**);
49. FFT(B,n,**false**);
50. **for**(**int** i=0;i<n;i++) A[i]=(**int**)((**long** **long**)A[i]\*B[i]%P);
51. FFT(A,n,**true**);
52. **return** 0;
53. }

## Fast Walsh-Hadamard Transform

### OR

1. **void** FWT(**int** \*P,**int** opt)
2. {
3. **for**(**int** i=2;i<=N;i<<=1)
4. **for**(**int** p=i>>1,j=0;j<N;j+=i)
5. **for**(**int** k=j;k<j+p;++k)
6. P[k+p]+=P[k]\*opt;
7. }

### AND

1. **void** FWT(**int** \*P,**int** opt)
2. {
3. **for**(**int** i=2;i<=N;i<<=1)
4. **for**(**int** p=i>>1,j=0;j<N;j+=i)
5. **for**(**int** k=j;k<j+p;++k)
6. P[k]+=P[k+p]\*opt;
7. }

### XOR

1. **void** FWT(**int** \*P,**int** opt)
2. {
3. **for**(**int** i=2;i<=N;i<<=1)
4. **for**(**int** p=i>>1,j=0;j<N;j+=i)
5. **for**(**int** k=j;k<j+p;++k)
6. {
7. **int** x=P[k],y=P[k+p];
8. P[k]=(x+y)%MOD;P[k+p]=(x-y+MOD)%MOD;
9. **if**(opt==-1)P[k]=1ll\*P[k]\*inv2%MOD,P[k+p]=1ll\*P[k+p]\*inv2%MOD;
10. }
11. }

## Matrix-Tree Theorem

1. #include <iostream>
2. #include <cstdio>
3. #include <cstring>
4. #include <algorithm>
5. #include <cmath>
6. **using** **namespace** std;
8. **const** **int** maxn =13;
9. **typedef** **long** **long** LL;
11. **int** degree[maxn];
12. LL C[maxn][maxn];
14. LL det(LL a[][maxn],**int** n){
15. LL ret=1;
16. **for**(**int** i=0;i<n;i++){
17. **for**(**int** j=i+1;j<n;j++){
18. **while**(a[j][i]){
19. LL t=a[i][i]/a[j][i];
20. **for**(**int** k=i;k<n;k++){
21. a[i][k]=a[i][k]-a[j][k]\*t;
22. }
24. **for**(**int** k=i;k<n;k++){
25. swap(a[i][k],a[j][k]);
26. }
27. ret=-ret;
28. }
29. }
31. **if**(a[i][i]==0){
32. **return** 0;
33. }
34. ret=ret\*a[i][i];
35. }
37. **if**(ret<0){
38. ret=-ret;
39. }
40. **return** ret;
41. }
43. **int** main(){
44. **int** t,n,m;
45. scanf("%d",&t);
46. **while**(t--){
47. memset(degree,0,**sizeof**(degree));
48. memset(C,0,**sizeof**(C));
49. scanf("%d%d",&n,&m);
50. **while**(m--){
51. **int** u,v;
52. scanf("%d%d",&u,&v);
53. u--;v--;
54. C[u][v]=C[v][u]=-1;
55. degree[v]++;
56. degree[u]++;
57. }
59. **for**(**int** i=0;i<n;i++){
60. C[i][i]=degree[i];
61. }
62. printf("%lld\n",det(C,n-1));
63. }
64. **return** 0;
65. }

## Chinese Remainder Theorem

1. //n个方程:x=a[i](mod m[i]) 0<=i<n
2. **typedef** **long** **long** LL;
3. LL CRT(**int** n,**int** \*a,**int** \*m)
4. {
5. LL M=1,d,y,x=0;
6. **for**(**int** i=0;i<n;i++) M\*=m[i];
7. **for**(**int** i=0;i<n;i++)
8. {
9. LL w=M/m[i];
10. gcd(m[i],w,d,d,y);
11. x=(x+y\*w\*a[i])%M;
12. }
13. **return** (x+M)%M;
14. }

## Miller-Rabin Test

1. **typedef** unsigned **long** **long** ull;
2. ull pri[13]={12,2,3,5,7,11,13,17,19,23,29,31,37};
3. **bool** Miller\_Rabin(ull X)
4. {
5. ull a,x,y,u,v;
6. **for**(ull i=1;i<=pri[0];i++)
7. {
8. **if**(X==pri[i]) **return** **true**;
9. **if**(!(X&pri[i]))
10. **return** **false**;
11. }
12. **for**(v=X-1,u=0;!(v%2);v>>=1,u++);
13. **for**(**int** T=1;T<=20;T++)
14. {
15. a=rand()%(x-2)+2;
16. x=pow(a,v,X);
17. **for**(ull i=1;i<=u;i++,x=y)
18. {
19. y=mUl(x,x,X);
20. **if**(y==1&&x!=1&&x!=X-1)
21. **return** **false**;
22. }
23. **if**(x!=1) **return** **false**;
24. }
25. **return** **true**;
26. }

# Gragh Theory

## Network-flow

### Dinic

1. **const** **int** MAXN=3050;
2. **const** **int** MAXM=4000050;
3. **const** **int** INF=1000000050;
4. **int** fir[MAXN],cur[MAXN],lev[MAXN],end[MAXM],next[MAXM],f[MAXM],ed,S,T;
5. **void** addedge(**int** u,**int** v,**int** cap)
6. {
7. end[++ed]=v;
8. next[ed]=fir[u];
9. fir[u]=ed;
10. f[ed]=cap;
11. end[++ed]=u;
12. next[ed]=fir[v];
13. fir[v]=ed;
14. f[ed]=0;
15. **return**;
16. }
17. **bool** BFS()
18. {
19. **int** u;
20. memset(lev,-1,**sizeof**(lev));
21. queue<**int**>q;
22. lev[S]=0;
23. q.push(S);
24. **while**(q.size())
25. {
26. u=q.front(); q.pop();
27. **for**(**int** i=fir[u];i;i=next[i]) **if**(f[i]&&lev[end[i]]==-1)
28. {
29. lev[end[i]]=lev[u]+1;
30. q.push(end[i]);
31. }
32. }
33. memcpy(cur,fir,**sizeof** fir);
34. **return** lev[T]!=-1;
35. }
36. **int** DFS(**int** u,**int** maxf)
37. {
38. **if**(u==T||!maxf) **return** maxf;
39. **int** cnt=0;
40. **for**(**int** &i=cur[u],tem;i;i=next[i]) **if**(f[i]&&lev[end[i]]==lev[u]+1)
41. {
42. tem=DFS(end[i],min(maxf,f[i]));
43. maxf-=tem;
44. f[i]-=tem;
45. f[i^1]+=tem;
46. cnt+=tem;
47. **if**(!maxf) **break**;
48. }
49. **if**(!cnt) lev[u]=-1;
50. **return** cnt;
51. }
52. **int** Dinic()
53. {
54. **int** ans=0;
55. **while**(BFS())
56. ans+=DFS(S,2147483647);
57. **return** ans;
58. }
59. **void** init(**int** SS,**int** TT)
60. {
61. memset(fir,0,**sizeof**(fir));
62. ed=1;
63. S=SS; T=TT;
64. **return**;
65. }

### Minimum Cost Maximum Flow

1. **const** **int** INF=0x7f7f7f7f;
2. **int** fir[MAXN],cur[MAXN],lev[MAXN],end[MAXM],next[MAXM],f[MAXM],mono[MAXM],ed,S,T;
3. **int** pre[MAXN];
4. **bool** exist[MAXN];
5. **void** init()
6. {
7. memset(fir,0,**sizeof**(fir));
8. ed=1; S=MAXN-2; T=MAXN-1;
9. **return**;
10. }
11. **void** addedge(**int** u,**int** v,**int** cap,**int** val)
12. {
13. end[++ed]=v;
14. next[ed]=fir[u];
15. fir[u]=ed;
16. f[ed]=cap;
17. mono[ed]=val;
18. end[++ed]=u;
19. next[ed]=fir[v];
20. fir[v]=ed;
21. f[ed]=0;
22. mono[ed]=-1\*val;
23. **return**;
24. }
25. **bool** BFS()
26. {
27. **int** u;
28. queue<**int**>q;
29. memset(exist,**false**,**sizeof**(exist));
30. memset(lev,127,**sizeof**(lev));
31. lev[S]=pre[S]=0;
32. q.push(S);
33. **while**(q.size())
34. {
35. u=q.front(); q.pop();
36. exist[u]=**false**;
37. **for**(**int** i=fir[u];i;i=next[i]) **if**(f[i]&&lev[u]+mono[i]<lev[end[i]])
38. {
39. lev[end[i]]=lev[u]+mono[i];
40. pre[end[i]]=i;
41. **if**(!exist[end[i]])
42. {
43. exist[end[i]]=**true**;
44. q.push(end[i]);
45. }
46. }
47. }
48. memcpy(cur,fir,**sizeof**(fir));
49. **return** lev[T]!=INF;
50. }
51. **int** DFS(**int** u,**int** maxf)
52. {
53. **if**(u==T||!maxf) **return** maxf;
54. exist[u]=**true**;
55. **int** cnt=0;
56. **for**(**int** &i=cur[u],tem;i;i=next[i]) **if**(f[i]&&lev[u]+mono[i]==lev[end[i]])
57. {
58. **if**(exist[end[i]]) **continue**;
59. tem=DFS(end[i],min(f[i],maxf));
60. maxf-=tem;
61. f[i]-=tem;
62. f[i^1]+=tem;
63. cnt+=tem;
64. **if**(!maxf) **break**;
65. }
66. **if**(!cnt) lev[u]=-1\*INF;
67. exist[u]=**false**;
68. **return** cnt;
69. }
70. **int** Augment()
71. {
72. **int** delta=INF;
73. **for**(**int** i=pre[T];i;i=pre[end[i^1]])
74. **if**(f[i]<delta)
75. delta=f[i];
76. **for**(**int** i=pre[T];i;i=pre[end[i^1]])
77. {
78. f[i]-=delta;
79. f[i^1]+=delta;
80. }
81. **return** delta\*lev[T];
82. }
83. **int** MCMF()
84. {
85. **int** ans=0;
86. memset(exist,**false**,**sizeof**(exist));
87. **while**(BFS())
88. //ans+=DFS(S,INF)\*lev[T];
89. ans+=Augment();
90. **return** ans;
91. }

### With Upper&Lower Bounds

#### 无源汇有上下界最大流

以前写的最大流默认的下界为0，而这里的下界却不为0，所以我们要进行再构造让每条边的下界为0，这样做是为了方便处理。对于每根管子有一个上界容量up和一个下界容量low，我们让这根管子的容量下界变为0，上界为up-low。可是这样做了的话流量就不守恒了，为了再次满足流量守恒，即每个节点"入流=出流”，我们增设一个超级源点st和一个超级终点sd。我们开设一个数组du[]来记录每个节点的流量情况。

du[i]=in[i]（i节点所有入流下界之和）-out[i]（i节点所有出流下界之和）。

当du[i]大于0的时候，st到i连一条流量为du[i]的边。

当du[i]小于0的时候，i到sd连一条流量为-du[i]的边。

最后对（st，sd）求一次最大流即可，当所有附加边全部满流时（即maxflow==所有du[]>0之和），有可行解。

#### 有源汇有上下界的最大流

源点s，终点d。超级源点ss，超级终点dd。首先判断是否存在满足所有边上下界的可行流，方法可以转化成无源汇有上下界的可行流问题。怎么转换呢？

增设一条从d到s没有下界容量为无穷的边，那么原图就变成了一个无源汇的循环流图。接下来的事情一样，超级源点ss连i（du[i]>0），i连超级汇点（du[i]<0）,

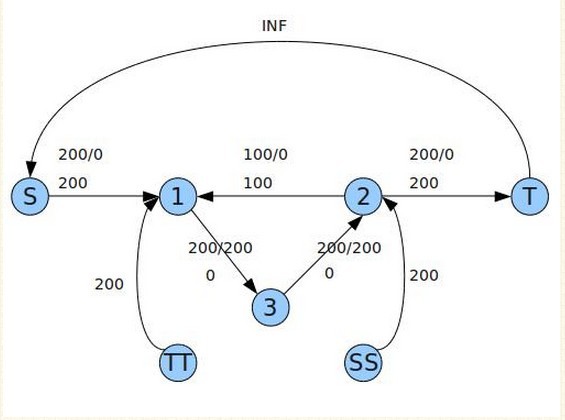
对（ss，dd）进行一次最大流，当maxflow等于所有(du[]>0)之和时，有可行流，否则没有。

当有可行流时，删除超级源点ss和超级终点dd，再对（s，d）进行一次最大流，此时得到的maxflow则为题目的解。为什么呢？因为第一次maxflow（）只是求得所有满足下界的流量，而残留网络（s，d）路上还有许多自由流（没有和超级源点和超级汇点连接的边）没有流满，所有最终得到的maxflow=（第一次流满下界的流+第二次能流通的自由流）。

#### 有源汇有上下界的最小流

建图模型：同样转换成先求无源汇有上下界的可行流，先添加一条d到s容量为无穷的边，这里求最小流很容易让人产生歧路，为什么呢？当所有边满足下界条件并且能量守恒时，这时候求得的最大流不就是最小流么。这样是错误了，我开始了在这揣测了良久。

下面来看个例子：



这样求得的最小流为200，而实际的可行最小流解只需100。

问题出在原图中存在环（循环流），而我们没有利用，导致流增大了。

解决方法：先不增加d->s容量为无穷的边，进行一次maxflow（），如果还没有满流，则加一条（d，s）容量为无穷的边，再进行一次maxflow（），当且仅当所有附加弧满载时，有可行解，解为flow[（d->s）^1]（即d到s的后悔边权值）。

## Link-Cut Tree

1. **const** **int** MAXN=10050;
2. **int** fa[MAXN],son[MAXN][2],st[MAXN];
3. **bool** rev[MAXN];
4. **bool** isroot(**int** x)
5. {// Check whether x is the root of the Auxiliary tree it belongs to
6. **return** x!=son[fa[x]][0]&&x!=son[fa[x]][1];
7. }
8. **void** Down(**int** x)
9. {
10. **if**(!rev[x]) **return**;
11. swap(son[son[x][0]][0],son[son[x][0]][1]);
12. swap(son[son[x][1]][0],son[son[x][1]][1]);
13. rev[son[x][0]]^=**true**;
14. rev[son[x][1]]^=**true**;
15. rev[x]=**false**;
16. **return**;
17. }
18. **void** adjust(**int** x)
19. {
20. **int** fa1=fa[x],fa2=fa[fa[x]],d;
21. d=son[fa1][1]==x;
22. **if**(!isroot(fa1))
23. son[fa2][son[fa2][1]==fa1]=x;
24. fa[x]=fa2;
25. fa[fa1]=x;
26. fa[son[x][d^1]]=fa1;
27. son[fa1][d]=son[x][d^1];
28. son[x][d^1]=fa1;
29. **return**;
30. }
31. **void** Splay(**int** x)
32. {
33. st[0]=0;
34. st[++st[0]]=x;
35. **for**(**int** i=x;!isroot(i);i=fa[i])
36. st[++st[0]]=fa[i];
37. **while**(st[0]) Down(st[st[0]--]);
38. **int** fa1,fa2;
39. **while**(!isroot(x))
40. {
41. fa1=fa[x];
42. fa2=fa[fa[x]];
43. **if**(!isroot(fa1))
44. {
45. **if**((son[fa1][1]==x)==(son[fa2][1]==fa1))
46. adjust(fa1);
47. **else** adjust(x);
48. }
49. adjust(x);
50. }
51. **return**;
52. }
53. **void** Access(**int** x)
54. {
55. **int** t=0;
56. **while**(x)
57. {
58. Splay(x);
59. son[x][1]=t;
60. t=x;
61. x=fa[x];
62. }
63. **return**;
64. }
65. **void** Change\_to\_root(**int** x)
66. {
67. Access(x); Splay(x);
68. swap(son[x][0],son[x][1]);
69. rev[x]^=**true**;
70. **return**;
71. }
72. **void** Link(**int** x,**int** y)
73. {
74. Change\_to\_root(y);
75. fa[y]=x;
76. //Splay(y);
77. **return**;
78. }
79. **void** Cut(**int** x,**int** y)
80. {
81. Change\_to\_root(x);
82. Access(y);//
83. Splay(y);
84. son[y][0]=fa[x]=0;
85. **return**;
86. }
87. **int** Find\_root(**int** x)
88. {
89. Access(x); Splay(x);
90. **while**(son[x][0]) x=son[x][0];
91. **return** x;
92. }
93. **void** read1n(**int** &x)
94. {
95. **char** ch=getchar();
96. **while**(ch<'0'||'9'<ch) ch=getchar();
97. **for**(x=0;'0'<=ch&&ch<='9';ch=getchar())
98. x=(x<<1)+(x<<3)+ch-'0';
99. **return**;
100. }

# Computational Geometry

1. **const** **double** eps=1e-10;
2. **const** **double** Pi=acos(-1.);
3. **int** dcmp(**const** **double** &x) { **if**(fabs(x)<eps) **return** 0; **return** x<0?-1:1; }
4. **typedef** **struct** Poi Vec;
5. **struct** Poi
6. {
7. **double** x,y;
8. Poi(**const** **double** &a=0,**const** **double** &b=0):x(a),y(b) { }
9. Poi operator+(**const** Poi &P)**const** { **return** Poi(x+P.x,y+P.y); }
10. Poi operator-(**const** Poi &P)**const** { **return** Poi(x-P.x,y-P.y); }
11. Poi operator\*(**const** **double** &P)**const** { **return** Poi(x\*P,y\*P); }
12. Poi operator/(**const** **double** &P)**const** { **return** Poi(x/P,y/P); }
13. **bool** operator<(**const** Poi &P)**const** { **return** !dcmp(x-P.x)?dcmp(y-P.y)<=0:dcmp(x-P.x)<=0; }
14. };
15. **struct** Line
16. {
17. Poi Ps; Vec Dir;
18. **double** ang;
19. Line(**const** Poi &ps=Poi(),**const** Vec &dir=Vec()):Ps(ps),Dir(dir)
20. { ang=atan2(Dir.y,Dir.x); }
21. **bool** operator<(**const** Line &L1)**const** { **return** ang<L1.ang; }
22. };
23. **struct** Circle
24. {
25. Poi Pc;
26. **double** r;
27. Circle(**const** Poi &P=Poi(),**const** **double** &R=0):Pc(P),r(R) { }
28. Poi loc(**double** ang,**double** offset) { **return** Poi(Pc.x+cos(ang)\*(r+offset),Pc.y+sin(ang)\*(r+offset)); }
29. };
30. **double** dOt(**const** Vec &V1,**const** Vec &V2) { **return** V1.x\*V2.x+V1.y\*V2.y; }
31. **double** cRoss(**const** Vec &V1,**const** Vec &V2) { **return** V1.x\*V2.y-V1.y\*V2.x; }
32. **double** lEnth(**const** Vec V) { **return** sqrt(dOt(V,V)); }
33. Vec rOtate(**const** Vec &V,**const** **double** &P)
34. { **double** cOs=cos(P),sIn=sin(P); **return** Vec(V.x\*cOs-V.y\*sIn,V.x\*sIn+V.y\*cOs); }
35. **double** aNgle(**const** Vec &V1,**const** Vec &V2) { **return** acos(dOt(V1,V2)/lEnth(V1)/lEnth(V2)); }
36. Poi iNtersect(**const** Poi &P1,**const** Vec &V1,**const** Poi &P2,**const** Vec &V2)
37. { **return** P1+V1\*(cRoss(V2,P1-P2)/cRoss(V1,V2)); }
38. Poi iNtersect(**const** Line &L1,**const** Line &L2)
39. { **return** iNtersect(L1.Ps,L1.Dir,L2.Ps,L2.Dir); }
40. **int** HPI(Line \*L,**int** N,Poi \*Pol)
41. {
42. **int** l,r,m;
43. Poi IP[N+50]; Line q[N+50];
44. sort(L,L+N);//按极角排序
45. q[l=r=0]=L[0];
46. **for**(**int** i=1;i<N;i++)
47. {
48. **while**(l<r&&dcmp(cRoss(L[i].Dir,IP[r-1]-L[i].Ps))<=0) r--; //新加入的直线可能是尾部的一些交点失效
49. **while**(l<r&&dcmp(cRoss(L[i].Dir,IP[l]-L[i].Ps))<=0) l++; //首部
50. q[++r]=L[i]; //加入
51. **if**(!dcmp(cRoss(q[r].Dir,q[r-1].Dir)))
52. { //对于平行直线要取靠左的
53. r--;
54. **if**(dcmp(cRoss(q[r+1].Dir,q[r].Ps-q[r+1].Ps))<0) q[r]=q[r+1];
55. }
56. **if**(l<r) IP[r-1]=iNtersect(q[r-1],q[r]); //如果队列中有至少两条线，则取交点
57. }
58. **while**(l<r&&dcmp(cRoss(q[l].Dir,IP[r-1]-q[l].Ps))<=0) r--; //后面一些交点可能实际上是无用的
59. **if**(r-l<2) **return** 0; //如果只有不到两条线，则失败了
60. IP[r]=iNtersect(q[l],q[r]); //将最后一条线和第一条线交起来
61. **for**(**int** i=l;i<=r;i++) Pol[m++]=IP[i];
62. **return** m;
63. }
64. **int** Andrew(Poi \*A,**int** N,Poi \*B)
65. {
66. **int** m=0;
67. sort(A,A+N);
68. **for**(**int** i=0;i<N;B[m++]=A[i++])
69. **while**(m>1&&cRoss(B[m-1]-B[m-2],A[i]-B[m-2])<=0) m--;
70. **for**(**int** i=N-2,k=m;i>=0;B[m++]=A[i--])
71. **while**(m>k&&cRoss(B[m-1]-B[m-2],A[i]-B[m-2])<=0) m--;
72. **if**(N>1) m--;
73. B[m]=B[0];
74. **return** m;
75. }
76. **int** ShamoS(**int** N,Poi \*P)
77. {
78. **int** r=1,ans=0,d;
79. **if**(N==1) **return** 0;
80. **else** **if**(N==2) **return** sQr(P[0]-P[1]);
81. **for**(**int** l=0;l<N;r=(r+1)%N)
82. {
83. d=dcmp(cRoss(P[l+1]-P[l],P[r+1]-P[r]));
84. **if**(d>0) **continue**;
85. ans=max(ans,(**int**)sQr(P[r--]-P[l++]));
86. **if**(!d) ans=max(ans,(**int**)sQr(P[r+1]-P[l]));
87. }
88. **return** ans;
89. }
90. **bool** onseg(**const** Poi &A,**const** Poi &B,**const** Poi &P)
91. {
92. Vec PA=A-P,PB=B-P;
93. **if**(!dcmp(cRoss(PA,PB))&&dcmp(dOt(PA,PB))<=0) **return** **true**;
94. **return** **false**;
95. }
96. **bool** segI(**const** Poi &A,**const** Poi &B,**const** Poi &C,**const** Poi &D)
97. {
98. **if**(onseg(A,B,C) ||onseg(A,B,D) ||onseg(C,D,A) ||onseg(C,D,B)) **return** **true**;
99. **if**((dcmp(cRoss(B-C,D-C))^dcmp(cRoss(A-C,D-C)))==-2
100. &&(dcmp(cRoss(D-B,A-B))^dcmp(cRoss(C-B,A-B)))==-2) **return** **true**;
101. **return** **false**;
102. }

# Divide&Conquer Algorithm

## Danqi Chen’s Algorithm

1. #include<iostream>
2. #include<cstdio>
3. #include<cstdlib>
4. #include<algorithm>
5. #include<cstring>
6. #define inf 0x7fffffff
7. #define ll long long
8. **using** **namespace** std;
9. **inline** **int** read()
10. {
11. **int** x=0,f=1;**char** ch=getchar();
12. **while**(ch>'9'||ch<'0'){**if**(ch=='-')f=-1;ch=getchar();}
13. **while**(ch>='0'&&ch<='9'){x=x\*10+ch-'0';ch=getchar();}
14. **return** x\*f;
15. }
16. **int** n,m,t[200005],ans[100005];
17. **struct** data{**int** a,b,c,s,ans;}a[100005],p[100005];
18. **inline** **bool** cmp(data a,data b)
19. {
20. **if**(a.a==b.a&&a.b==b.b)**return** a.c<b.c;
21. **if**(a.a==b.a)**return** a.b<b.b;
22. **return** a.a<b.a;
23. }
24. **inline** **bool** operator<(data a,data b)
25. {
26. **if**(a.b==b.b)**return** a.c<b.c;
27. **return** a.b<b.b;
28. }
29. **inline** **int** lowbit(**int** x){**return** x&(-x);}
30. **inline** **void** update(**int** x,**int** num)
31. {
32. **for**(**int** i=x;i<=m;i+=lowbit(i))
33. t[i]+=num;
34. }
35. **inline** **int** ask(**int** x)
36. {
37. **int** tmp=0;
38. **for**(**int** i=x;i;i-=lowbit(i))
39. tmp+=t[i];
40. **return** tmp;
41. }
42. **void** solve(**int** l,**int** r)
43. {
44. **if**(l==r)**return**;
45. **int** mid=(l+r)>>1;
46. solve(l,mid);solve(mid+1,r);
47. sort(p+l,p+mid+1);
48. sort(p+mid+1,p+r+1);
49. **int** i=l,j=mid+1;
50. **while**(j<=r)
51. {
52. **while**(i<=mid&&p[i].b<=p[j].b)
53. {
54. update(p[i].c,p[i].s);
55. i++;
56. }
57. p[j].ans+=ask(p[j].c);
58. j++;
59. }
60. **for**(**int** j=l;j<i;j++)update(p[j].c,-p[j].s);
61. }
62. **int** main()
63. {
64. **int** N=read();m=read();
65. **for**(**int** i=1;i<=N;i++)
66. a[i].a=read(),a[i].b=read(),a[i].c=read();
67. sort(a+1,a+N+1,cmp);
68. **int** cnt=0;
69. **for**(**int** i=1;i<=N;i++)
70. {
71. cnt++;
72. **if**(a[i].a!=a[i+1].a||a[i].b!=a[i+1].b||a[i].c!=a[i+1].c)
73. {
74. p[++n]=a[i];
75. p[n].s=cnt;
76. cnt=0;
77. }
78. }
79. solve(1,n);
80. **for**(**int** i=1;i<=n;i++)
81. ans[p[i].ans+p[i].s-1]+=p[i].s;
82. **for**(**int** i=0;i<N;i++)
83. printf("%d\n",ans[i]);
84. **return** 0;
85. }

## D&C Algorithm on Tree

### Divide by Chains

### Divide by Points

# Partition

## Block Array

## Baby-Step-Giant-Step

# Other Algorithm

## 奇技淫巧

1. **inline** **int** mul\_mod0(**int** a,**int** b)
2. {
3. **int** ret;
4. \_\_asm\_\_ \_\_volatile\_\_("\tmull %%ebx\n\tdivl %%ecx\n":"=d"(ret):"a"(a),"b"(b),"c"(mod));
5. **return** ret;
6. }
7. **inline** **long** **long** mul\_mod1(**long** **long** a,**long** **long** b)
8. {
9. **long** **long** d=(**long** **long**)floor(a\*(**double**)b/mod+0.5);
10. **long** **long** ans=a\*b-d\*mod;
11. **if**(ans<0) ans+=mod;
12. **return** ans;
13. }
14. **inline** **long** **long** mul\_mod2(**long** **long** a,**long** **long** b)
15. {
16. //https://blog.csdn.net/xuxiayang/article/details/81623511
17. a%=mod; b%=mod;
18. **long** **double** c=a\*b/mod;
19. **long** **long** ans=a\*b-c\*mod;
20. **if**(ans<0) ans+=mod;
21. **if**(mod<=ans) ans-=mod;
22. **return** ans;
23. }
24. **void** calc\_inv()
25. {
26. inv[1]=1;
27. **for**(**int** i=2;i<MAXN;i++)
28. {
29. inv[i]=mul\_mod(mod-mod/i,inv[mod%i],mod);
30. cerr<<mul\_mod(inv[i],i,mod)<<endl;
31. }
32. **return**;
33. }
34. **for**(**int** i=S;i;i=(i-1)&S)
35. {//枚举S的子集
36. ...
37. }
38. **for**(**int** S=(1<<k)-1;s<1<<n;)
39. {//枚举一个大小为n的集合大小为k的子集
40. ...
41. **int** x=S&-S;
42. **int** y=S+x;
43. S=((S&~y)/x>>1)|y;
44. }

## Min25筛

1. **using** **namespace** std;
2. #define pb push\_back
3. #define mp make\_pair
4. **typedef** pair<**int**,**int**> pii;
5. **typedef** **long** **long** ll;
6. **typedef** **double** ld;
7. **typedef** vector<**int**> vi;
8. #define fi first
9. #define se second
10. #define fe first
11. #define FO(x) {freopen(#x".in","r",stdin);freopen(#x".out","w",stdout);}
12. #define Edg int M=0,fst[SZ],vb[SZ],nxt[SZ];void ad\_de(int a,int b){++M;nxt[M]=fst[a];fst[a]=M;vb[M]=b;}void adde(int a,int b){ad\_de(a,b);ad\_de(b,a);}
13. #define Edgc int M=0,fst[SZ],vb[SZ],nxt[SZ],vc[SZ];void ad\_de(int a,int b,int c){++M;nxt[M]=fst[a];fst[a]=M;vb[M]=b;vc[M]=c;}void adde(int a,int b,int c){ad\_de(a,b,c);ad\_de(b,a,c);}
14. #define es(x,e) (int e=fst[x];e;e=nxt[e])
15. #define esb(x,e,b) (int e=fst[x],b=vb[e];e;e=nxt[e],b=vb[e])
16. **typedef** unsigned us;
17. **typedef** unsigned **long** **long** ull;
18. **const** us MOD=1e9+7;
19. #define SS 2333333
20. ll n,c0[SS],c1[SS],b0[SS],b1[SS]; **int** S,ps[SS/10],pn=0;
21. **inline** ull F(ull x,us g)
22. {
23. **if**(x<=1||ps[g]>x) **return** 0;
24. ull ans=((x>S)?b1[n/x]:c1[x])-c1[ps[g-1]]+MOD; //注意这里原来有bug
25. **for**(us j=g;j<=pn&&ps[j]\*(ll)ps[j]<=x;++j)
26. {
27. ull cn=x/ps[j],ce=ps[j]\*(ll)ps[j];
28. **for**(us e=1;cn>=ps[j];++e,cn/=ps[j],ce\*=ps[j])
29. ans+=F(cn,j+1)\*(ps[j]^e)+(ps[j]^(e+1)),ans%=MOD;
30. }
31. **return** ans%MOD;
32. }
33. **int** main()
34. {
35. cin>>n; S=sqrtl(n);
36. **for**(**int** i=1;i<=S;++i)
37. {
38. ll t=(n/i)%MOD; b0[i]=t;
39. b1[i]=t\*(t+1)/2%MOD; c0[i]=i;
40. c1[i]=i\*(ll)(i+1)/2%MOD;
41. }
42. **for**(**int** i=2;i<=S;++i)
43. {
44. **if**(c0[i]==c0[i-1]) **continue**; //not a prime
45. ll x0=c0[i-1],x1=c1[i-1],r=(ll)i\*i; ps[++pn]=i;
46. **int** u=min((ll)S,n/(i\*(ll)i)),uu=min(u,S/i);
47. **for**(**int** j=1;j<=uu;++j)
48. b1[j]-=(b1[j\*i]-x1)\*i,
49. b0[j]-=b0[j\*i]-x0;
50. ll t=n/i;
51. **if**(t<=2147483647)
52. {
53. **int** tt=t;
54. **for**(**int** j=uu+1;j<=u;++j)
55. b1[j]-=(c1[tt/j]-x1)\*i,
56. b0[j]-=c0[tt/j]-x0;
57. }
58. **else**
59. {
60. **for**(**int** j=uu+1;j<=u;++j)
61. b1[j]-=(c1[t/j]-x1)\*i,
62. b0[j]-=c0[t/j]-x0;
63. }
64. **for**(**int** j=S;j>=r;--j)
65. c1[j]-=(c1[j/i]-x1)\*i,
66. c0[j]-=c0[j/i]-x0;
67. }
68. **for**(**int** i=1;i<=S;++i)
69. {
70. c1[i]-=c0[i];
71. b1[i]-=b0[i];
72. **if**(i>=2) c1[i]+=2;
73. **if**(n>=2LL\*i) b1[i]+=2;
74. c1[i]=(c1[i]%MOD+MOD)%MOD;
75. b1[i]=(b1[i]%MOD+MOD)%MOD;
76. }
77. ps[pn+1]=S+1;
78. ll ans=1+F(n,1);
79. ans=(ans%MOD+MOD)%MOD;
80. printf("%d\n",**int**(ans));
81. }