代码仓库

目录：

01.【数学方法】矩阵快速幂

02.【数学方法】高斯消元（naïve版）

03.【数学方法】高斯消元（mid版）

04.【字符串啊】Manacher算法（回文串）

05.【字符串啊】KMP（字符串匹配）

06.【数据结构】线段树（ZKW单点修改）

07.【数据结构】线段树（RMQ）

08.【数据结构】线段树（区间加+赋值）

09.【数据结构】Splay树（未完全测试）

10.【数据结构】AVL树（平衡树）

11.【图论图论】最小生成树（prim）

12.【图论图论】次小生成树

13.【图论图论】最大流（Dinic）

14.【图论图论】LCA+最大生成树（truck）

15.【动态规划】背包01，多重，完全

矩阵模板

|  |
| --- |
| #include<cstdio>  #include<cstring>  #include<iostream>  **using** **namespace** std**;**  **typedef** long long ll**;**  const int P **=** 10000007**;**  const int N**=**13**;**  ll n**,**m**;**  struct matrix**{**  ll a**[**N**][**N**];**  int row**,**col**;**  matrix**():**row**(**N**),**col**(**N**){**memset**(**a**,**0**,sizeof(**a**));}**  matrix**(**int x**,**int y**):**row**(**x**),**col**(**y**){**memset**(**a**,**0**,sizeof(**a**));}**  ll**\*** **operator** **[]** **(**int x**){return** a**[**x**];}**  matrix **operator** **\*** **(**matrix x**){**  matrix tmp **;**  **for** **(**int i**=**0**;**i**<=**n**+**1**;**i**++)**  **for** **(**int j**=**0**;**j**<=**n**+**1**;**j**++){**  tmp**[**i**][**j**]=**0**;**  **for** **(**int k**=**0**;**k**<=**n**+**1**;**k**++)**  tmp**[**i**][**j**]=(**tmp**[**i**][**j**]+**a**[**i**][**k**]\***x**[**k**][**j**])%**P**;**  **}**  **return** tmp**;**  **}**  void **operator** **\*=** **(**matrix x**){\*this** **=** **\*this** **\*** x**;}**  matrix **operator** **^** **(**ll x**){**  matrix ret**;**  **for** **(**int i**=**0**;**i**<=**n**+**1**;**i**++)**ret**[**i**][**i**]=**1**;**  matrix tmp **=** **\*this;**  **for** **(;**x**;**x**>>=**1**,**tmp**\*=**tmp**){if(**x**&**1**)**ret **\*=**tmp**;}**  **return** ret**;**  **}**  void print**(){**  **for** **(**int i**=**0**;**i**<=**n**+**1**;**i**++){**  **for** **(**int j**=**0**;**j**<=**n**+**1**;**j**++)**  printf**(**"%d "**,**a**[**i**][**j**]);**  puts**(**""**);**  **}**  **}**  **};** |

高斯消元，判断有无解的

|  |
| --- |
| #include<cstdio>  #include<cmath>  #include<cstring>  #include<iostream>  #include<vector>  **using** **namespace** std**;**  **typedef** long long LL**;**  const double EPS**=**1e-6**;**  const int N**=**55**;**  struct matrix**{**  int a**[**N**][**N**];**  int row**,**col**;**  matrix**():**row**(**N**),**col**(**N**){**memset**(**a**,**0**,sizeof(**a**));}**  matrix**(**int x**,**int y**):**row**(**x**),**col**(**y**){**  memset**(**a**,**0**,sizeof(**a**));**  **}**  int**\*** **operator** **[](**int x**){return** a**[**x**];}**  void print**(){**  **for** **(**int i**=**0**;**i**<**row**;**i**++){**  **for** **(**int j**=**0**;**j**<**col**;**j**++)**  printf**(**"%d "**,**a**[**i**][**j**]);**  puts**(**""**);**  **}**  puts**(**""**);**  **}**  **};**  int Gauss**(**matrix a**,**int m**,**int n**){**  int x\_cnt **=** 0**;**  int col**,** k**;** //col为列号,k为行号  **for** **(**k**=**0**,**col**=**0**;**k**<**m**&&**col**<**n**;** **++**k**,** **++**col**){**  int r **=** k**;** //r为第col列的一个1  **for** **(**int i**=**k**;**i**<**m**;++**i**)** **if** **(**a**[**i**][**col**])**r**=**i**;**  **if** **(!**a**[**r**][**col**]){** k**--;** **continue;}**  **if** **(**r**!=**k**)for** **(**int i**=**col**;**i**<=**n**;++**i**)**  swap**(** a**[**r**][**i**],** a**[**k**][**i**]);**  **for** **(**int i**=**k**+**1**;**i**<**m**;** **++**i**)if** **(**a**[**i**][**col**])**//消元  **for** **(**int j**=**col**;**j**<=**n**;++**j**)**a**[**i**][**j**]^=**a**[**k**][**j**];**  **}**  **for** **(**int i**=**k**;**i**<**m**;++**i**)** **if** **(**a**[**i**][**n**])return** **-**1**;**  **if** **(**k**<=**n**)return** n**-**k**;** //返回自由元个数  **}** |

高斯消元，求出一组解的

|  |
| --- |
| #include <iostream>  #include <algorithm>  #include <cstdio>  #include <cstring>  #include <cmath>  **using** **namespace** std**;**  const int N **=** 1010**;**  const double EPS**=**1e-7**;**  int m**,**n**;**  double a**[**N**][**N**],**x**[**N**];**  int Gauss**(**int m**,**int n**){**  int col**=**0**,** k**=**0**;**//col为列号,k为行号  **for** **(;**k**<**m**&&**col**<**n**;++**k**,++**col**){**  int r **=** k**;**  **for** **(**int i**=**k**+**1**;**i**<**m**;++**i**)**  **if(**fabs**(**a**[**i**][**col**])>**fabs**(**a**[**r**][**col**]))**r**=**i**;**  **if** **(**fabs**(**a**[**r**][**col**])<**EPS**){**k**--;continue;}**//列全为0  **if** **(**r**!=**k**)for(**int i**=**col**;**i**<=**n**;++**i**)**  swap**(**a**[**k**][**i**],**a**[**r**][**i**]);**  **for** **(**int i**=**k**+**1**;**i**<**m**;++**i**)**//消元  **if(**fabs**(**a**[**i**][**col**])>**EPS**){**  double t **=** a**[**i**][**col**]/**a**[**k**][**col**];**  **for** **(**int j**=**col**;**j**<=**n**;**j**++)**a**[**i**][**j**]-=**a**[**k**][**j**]\***t**;**  a**[**i**][**col**]** **=** 0**;**  **}**  **}**  **for(**int i**=**k **;**i**<**m **;++**i**)**//无解  **if** **(**fabs**(**a**[**i**][**n**])>**EPS**)** **return** **-**1**;**  **if** **(**k **<** n**)** **return** n **-** k**;** //自由元个数  **for** **(**int i **=**n**-**1**;** i**>=**0**;** i**--){**//回带求解  double temp **=** a**[**i**][**n**];**  **for** **(**int j**=**i**+**1**;** j**<**n**;** **++**j**)**  temp **-=** x**[**j**]** **\*** a**[**i**][**j**];**  x**[**i**]** **=** **(**temp **/** a**[**i**][**i**]);**  **}**  **return** 0**;**  **}** |

Manacher算法

|  |
| --- |
| #include<cstdio>  #include<string>  #include<cstring>  #include<iostream>  #include<algorithm>  **using** **namespace** std**;**  const int N**=**233333**;**//20W  //在o(n)时间内算出以每个点为中心的最大回文串长度  int Manacher**(**string st**){**  int len**=**st**.**size**();**  int **\***p**=new** int**[**len**+**1**];**  memset**(**p**,**0**,sizeof(**p**));**  int mx**=**0**,**id**=**0**;**  **for** **(**int i**=**1**;**i**<=**len**;**i**++){**  **if** **(**mx**>**i**)**p**[**i**]=**min**(**p**[**2**\***id**-**i**],**mx**-**i**);**  **else** p**[**i**]=**1**;**  **while** **(**st**[**i**+**p**[**i**]]==**st**[**i**-**p**[**i**]])**p**[**i**]++;**  **if** **(**i**+**p**[**i**]>**mx**){**mx**=**i**+**p**[**i**];**id**=**i**;}**  **}**  int ma**=**0**;**  **for(**int i**=**1**;**i**<**len**;**i**++)**ma**=**max**(**ma**,**p**[**i**]);**  **delete(**p**);**  **return** ma**-**1**;**  **}**  int main**(){**  //freopen("fuck.in","r",stdin);  char st**[**N**];**  **while** **(~**scanf**(**"%s"**,**st**)){**  string st0**=**"$#"**;**  **for** **(**int i**=**0**;**st**[**i**]!=**'\0'**;**i**++){**  st0**+=**st**[**i**];** st0**+=**"#"**;**  **}**  printf**(**"%d\n"**,**Manacher**(**st0**));**  **}**  **return** 0**;**  **}** |

KMP 字符串匹配

|  |
| --- |
| #include<cstdio>  #include<cstring>  **using** **namespace** std**;**  **typedef** long long ll**;**  const int N**=**100007**;**  const int P**=**1000000007**;**  char a**[**N**],**b**[**N**];**  bool mat**[**N**];**  int next**[**N**];**  ll f**[**N**];**  void getNext**(**int m**){**  int i**=**0**,**j**=-**1**;**  next**[**0**]=-**1**;**  **while** **(**i**<**m**){**  **if** **(**j**==-**1**||**b**[**i**]==**b**[**j**]){**  **if** **(**b**[++**i**]!=**b**[++**j**])**next**[**i**]=**j**;**  **else** next**[**i**]=**next**[**j**];**  **}else** j**=**next**[**j**];**  **}**  **}**  void KMP**(**int n**,**int m**){**  memset**(**mat**,**0**,sizeof(**mat**));**  int i**=**0**,**j**=**0**;**  getNext**(**m**);**  **while** **(**i**<**n**&&**j**<**m**){**  **if** **(**j**==-**1**||**a**[**i**]==**b**[**j**])**i**++,**j**++;**  **else** j**=**next**[**j**];**  **if** **(!**i**&&!**j**)break;**  **if** **(**j**==**m**){**  mat**[**i**]=**1**;**  //printf("mat[%d]get\n",i);  j**=**next**[**j**];**  **}**  **}**  **}** |

线段树(ZKW大法)

|  |
| --- |
| #include<cstdio>  #include<cstring>  #include<iostream>  #include<algorithm>  **using** **namespace** std**;**  const int INF**=**0x3f3f3f3f**;**  const int N**=**3000100**;**  struct linetree**{**  #define lc (t<<1)  #define rc (t<<1^1)  int mi**[**N**],**M**;**  inline void build**(**int n**){**  M**=**1**;** **while(**M**<**n**)**M**<<=**1**;** M**--;**  memset**(**mi**,**INF**,sizeof(**mi**));**  **for** **(**int i**=**1**+**M**;**i**<=**n**+**M**;**i**++)** scanf**(**"%d"**,** **&**mi**[**i**]);**  **for** **(**int t**=**M**;**t**>=**1**;**t**--)**mi**[**t**]=**min**(**mi**[**lc**],**mi**[**rc**]);**  **}**  void change**(**int t**,**int x**){**  **for** **(**mi**[**t**+=**M**]=**x**,**t**>>=**1**;**t**;**t**>>=**1**)**  mi**[**t**]=**min**(**mi**[**lc**],**mi**[**rc**]);**  **}**  int query**(**int l**,**int r**){**  int ans **=** INF**;**  **for** **(**l**+=**M**-**1**,**r**+=**M**+**1**;**l**^**r**^**1**;**l**>>=**1**,**r**>>=**1**){**  **if** **(~**l**&**1**)**ans**=**min**(**ans**,**mi**[**l**^**1**]);**  **if** **(** r**&**1**)**ans**=**min**(**ans**,**mi**[**r**^**1**]);**  **}**  **return** ans**;**  **}**  **}**T**;**  int main**(){**  int n**,**q**,**ord**,**x**,**y**;**  **for** **(;~**scanf**(**"%d"**,&**n**);){**  T**.**build**(**n**);**  **for** **(**scanf**(**"%d"**,&**q**);**q**--;){**  scanf**(**"%d%d%d"**,&**ord**,&**x**,&**y**);**  **if** **(**ord**)**T**.**change**(**x**,**y**);**  **else** printf**(**"%d\n"**,**T**.**query**(**x**,**y**));**  **}**  **}**  **return** 0**;**  **}** |

线段树（RMQ）

|  |
| --- |
| #include<cstdio>  #include<cstring>  #include<iostream>  #include<algorithm>  **using** **namespace** std**;**  const int INF**=**0x3f3f3f3f**;**  const int N**=**600100**;**  int n**,**ans**,**m**,**a**[**N**];**  struct node **{**  int l**,**r**,**id**;**  node **()** **{** **}**  node**(**int x**,**int y**,**int z**){**l**=**x**;**r**=**y**;**id**=**z**;}**  **}**b**[**N**],**c**[**N**];**  inline bool cmp1**(**node a**,**node b**){return** a**.**l**<**b**.**l**;}**  inline bool cmp2**(**node a**,**node b**){return** a**.**r**<**b**.**r**;}**  struct linetree**{**  #define lc (t<<1)  #define rc (t<<1^1)  #define mid (l[t]+r[t]>>1)  int l**[**N**],**r**[**N**],**ma**[**N**],**mi**[**N**],**M**,**ta**[**N**],**ti**[**N**];**  inline void build**(**int n**){**  M**=**1**;** **while(**M**<**n**)**M**<<=**1**;** M**--;**  memset**(**ma**,** 0 **,sizeof(**ma**));**  memset**(**mi**,**INF**,sizeof(**mi**));**  memset**(**ta**,** 0 **,sizeof(**ta**));**  memset**(**ti**,**INF**,sizeof(**ti**));**  **for** **(**int i**=**1**+**M**;**i**<=**M**\***2**+**1**;**i**++)**l**[**i**]=**r**[**i**]=** i**-**M **;**  **for** **(**int t**=**M**;**t**>=**1**;**t**--)**l**[**t**]=**l**[**lc**],**r**[**t**]=**r**[**rc**];**  **}**  inline void down**(**int t**){**  **if** **(**t**>**M**)return** **;**//leaf node  ma**[**lc**]=**max**(**ma**[**lc**],**ta**[**t**]);**  ma**[**rc**]=**max**(**ma**[**rc**],**ta**[**t**]);**  ta**[**lc**]=**max**(**ta**[**lc**],**ta**[**t**]);**  ta**[**rc**]=**max**(**ta**[**rc**],**ta**[**t**]);**  ta**[**t**]** **=** 0**;**  mi**[**lc**]=**min**(**mi**[**lc**],**ti**[**t**]);**  mi**[**rc**]=**min**(**mi**[**rc**],**ti**[**t**]);**  ti**[**lc**]=**min**(**ti**[**lc**],**ti**[**t**]);**  ti**[**rc**]=**min**(**ti**[**rc**],**ti**[**t**]);**  ti**[**t**]** **=** INF**;**  **}**  inline void maintain**(**int t**){**  ma**[**t**]=**max**(**ma**[**lc**],**ma**[**rc**]);**  mi**[**t**]=**min**(**mi**[**lc**],**mi**[**rc**]);**  **}**  inline void tag**(**int t**,**int x**){**  ma**[**t**]=**max**(**ma**[**t**],**x**);**  mi**[**t**]=**min**(**mi**[**t**],**x**);**  ta**[**t**]=**max**(**ta**[**t**],**x**);**  ti**[**t**]=**min**(**ti**[**t**],**x**);**  **}**  void change**(**int t**,**int L**,**int R**,**int x**){**  **if** **(**L**<=**l**[**t**]&&**r**[**t**]<=**R**){**tag**(**t**,**x**);return;}**//in  down**(**t**);**  **if** **(**L**<=**mid**)**change**(**lc**,**L**,**R**,**x**);**  **if** **(**mid**<** R**)**change**(**rc**,**L**,**R**,**x**);**  maintain**(**t**);**  **}**  void query**(**int t**){**  **if** **(**t**>**M**){**//leaf node  b**[**t**-**M**]=**c**[**t**-**M**]=**node**(**mi**[**t**],**ma**[**t**],**t**-**M**);**  **return** **;**  **}**  down**(**t**);**  query**(**lc**);**  query**(**rc**);**  maintain**(**t**);**  **}**  **}**T**;** |

线段树（区间加+赋值）

|  |
| --- |
| #include<cstdio>  #include<cstring>  #include<iostream>  #include<algorithm>  **using** **namespace** std**;**  const int N **=**260000**;**  int n**,**m**;**  struct linetree**{**  #define lc (t<<1)  #define rc (t<<1^1)  #define mid (l[t]+r[t]>>1)  int l**[**N**],**r**[**N**],**M**,**tag**[**N**],**sum**[**N**],**len**[**N**],**Set**[**N**];**  inline void build**(**int n**){**  M**=**1**;** **while(**M**<**n**)**M**<<=**1**;** M**--;**//get M  memset**(**sum**,**0**,sizeof(**sum**));**  memset**(**tag**,**0**,sizeof(**tag**));**  memset**(**Set**,**0**,sizeof(**Set**));**  **for** **(**int i**=**1**+**M**;**i**<=**M**\***2**+**1**;**i**++){**//leaf  **if(**i**<=**n**+**M**)**scanf**(**"%d"**,&**sum**[**i**]);**  l**[**i**]** **=** r**[**i**]** **=** i**-**M **;**  len**[**i**]=**1**;**  **}**  **for** **(**int t**=**M**;**t**>=**1**;**t**--){**//fathers  sum**[**t**]=**sum**[**lc**]+**sum**[**rc**];**  l**[**t**]=**l**[**lc**],** r**[**t**]=**r**[**rc**];**  len**[**t**]=**len**[**lc**]+**len**[**rc**];**  **}**  **}**  inline void down**(**int t**){**  **if** **(**t**>**M**){**Set**[**t**]=**tag**[**t**]=**0**;return** **;}**//leaf node  **if** **(**Set**[**t**]){**  sum**[**lc**]=**Set**[**t**]\***len**[**lc**];**  sum**[**rc**]=**Set**[**t**]\***len**[**rc**];**  Set**[**lc**]=**Set**[**t**];**  Set**[**rc**]=**Set**[**t**];**  Set**[**t**]** **=** 0**;**  tag**[**lc**]=**tag**[**rc**]=**0**;**  **}**  **if** **(**tag**[**t**]){**  sum**[**lc**]+=**tag**[**t**]\***len**[**lc**];**  sum**[**rc**]+=**tag**[**t**]\***len**[**rc**];**  tag**[**lc**]+=**tag**[**t**];**  tag**[**rc**]+=**tag**[**t**];**  tag**[**t**]** **=** 0**;**  **}**  **}**  inline void \_tag**(**int t**,**int x**){**  sum**[**t**]+=**x**\***len**[**t**];**  tag**[**t**]+=**x**;**  **}**  inline void \_set**(**int t**,**int x**){**  sum**[**t**]=**x**\***len**[**t**];**  Set**[**t**]=**x**;**  tag**[**t**]=**0**;**  **}**  void change**(**int t**,**int L**,**int R**,**int x**){**  **if** **(**L**<=**l**[**t**]&&**r**[**t**]<=**R**){**\_tag**(**t**,**x**);return;}**  down**(**t**);**  **if** **(**L**<=**mid**)**change**(**lc**,**L**,**R**,**x**);**  **if** **(**mid**<** R**)**change**(**rc**,**L**,**R**,**x**);**  sum**[**t**]=**sum**[**lc**]+**sum**[**rc**];**  **}**  void set**(**int t**,**int L**,**int R**,**int x**){**  **if** **(**L**<=**l**[**t**]&&**r**[**t**]<=**R**){**\_set**(**t**,**x**);return;}**//in  down**(**t**);**  **if** **(**L**<=**mid**)**set**(**lc**,**L**,**R**,**x**);**  **if** **(**mid**<** R**)**set**(**rc**,**L**,**R**,**x**);**  sum**[**t**]=**sum**[**lc**]+**sum**[**rc**];**  **}**  int query**(**int t**,**int L**,**int R**){**  **if** **(**L**<=**l**[**t**]&&**r**[**t**]<=**R**)return** sum**[**t**];**  down**(**t**);**  int ans **=** 0**;**  **if** **(**L**<=**mid**)**ans**+=**query**(**lc**,**L**,**R**);**  **if** **(**mid**<** R**)**ans**+=**query**(**rc**,**L**,**R**);**  **return** ans**;**  **}**  **}**T**;** |

Splay-Tree

|  |
| --- |
| #include<cstdio>  #include<algorithm>  **using** **namespace** std**;**  struct Node**{**  int key**;**//size  Node **\***l**,\***r**,\***f**;**//left,right,father  **};**  class SplayTree**{**  public**:**  void Init**(){**rt**=NULL;}**  void Zag**(**Node **\***x**){**//left rotate  Node **\***y**=**x**->**f**;**//y is the father of x  y**->**r **=** x**->**l**;**  **if** **(**x**->**l**)**x**->**l**->**f **=** y**;**//if x has left child  x**->**f **=**y**->**f**;**  **if** **(**y**->**f**){**//y is not root  **if** **(**y**==**y**->**f**->**l**)**y**->**f**->**l**=**x**;**//y if left child  **else** y**->**f**->**r**=**x**;**//y is right child  **}**  y**->**f**=**x**;** x**->**l**=**y**;**  **}**  void Zig**(**Node **\***x**){**//right rotate  Node **\***y**=**x**->**f**;**//y is the father of x  y**->**l **=** x**->**r**;**  **if** **(**x**->**r**)**x**->**r**->**f**=**y**;**  x**->**f **=** y**->**f**;**  **if** **(**y**->**f**){**  **if** **(**y**==**y**->**f**->**l**)**y**->**f**->**l**=**x**;**  **else** y**->**f**->**r**=**x**;**  **}**  y**->**f**=**x**;**  x**->**r**=**y**;**  **}**  void Splay**(**Node **\***x**){**  **while** **(**x**->**f**){**  Node **\***p**=**x**->**f**;**  **if** **(!**p**->**f**){**  **if** **(**x**==**p**->**l**)**Zig**(**x**);**  **else** Zag**(**x**);**  **}else** **if** **(**x**==**p**->**l**){**  **if** **(**p**==**p**->**f**->**l**){**Zig**(**p**);**Zig**(**x**);}**  **else** **{**Zig**(**x**);**Zag**(**x**);}**  **}else** **{**//x==p->r  **if** **(**p**==**p**->**f**->**r**){**Zag**(**p**);**Zag**(**x**);}**  **else** **{**Zag**(**x**);**Zig**(**x**);}**  **}**  **}**  rt**=**x**;**  **}**  Node **\***Find**(**int x**){**  Node **\***T**=**rt**;**  **while** **(**T**){**  **if** **(**T**->**key**==**x**){**Splay**(**T**);return** T**;}**  **else** **if** **(**x**<**T**->**key**)**T**=**T**->**l**;**  **else** T**=**T**->**r**;**  **}**  **return** T**;**  **}**  void Insert**(**int x**){**  Node **\***T**=**rt**,\***fa**=NULL;**  **while** **(**T**){**  fa**=**T**;**  **if** **(**x**<**T**->**key**)**T**=**T**->**l**;**  **else** **if(**x**>**T**->**key**)**T**=**T**->**r**;**  **else** **return** **;**//two the same keys  **}**  T**=(**Node**\*)**malloc**(sizeof(**Node**));**  T**->**key**=**x**;**  T**->**l**=**T**->**r**=NULL;**  T**->**f**=**fa**;**  **if** **(**fa**){**  **if** **(**fa**->**key**>**x**)**fa**->**l**=**T**;**  **else** fa**->**r**=**T**;**  **}**  Splay**(**T**);**  **}**  void Delete**(**int x**){**  Node **\***T**=**Find**(**x**);**  **if** **(NULL==**T**)return** **;**//error  rt**=**Join**(**T**->**l**,**T**->**r**);**  **}**  Node **\***Maxnum**(**Node **\***t**){**  Node **\***T**=**t**;**  **while** **(**T**->**r**)**T**=**T**->**r**;**  Splay**(**T**);**  **return** T**;**  **}**  Node **\***Minnum**(**Node **\***t**){**  Node **\***T**=**t**;**  **while** **(**T**->**l**)**T**=**T**->**l**;**  Splay**(**T**);**  **return** T**;**  **}**  Node **\***Last**(**int x**){**  Node **\***T**=**Find**(**x**);**  T**=**T**->**l**;**  **return** **(**Maxnum**(**T**));**  **}**  Node **\***Next**(**int x**){**  Node **\***T**=**Find**(**x**);**  T**=**T**->**r**;**  **return** **(**Minnum**(**T**));**  **}**  Node **\***Join**(**Node **\***t1**,**Node **\***t2**){**  **if** **(NULL==**t1**)return** t2**;**  **if** **(NULL==**t2**)return** t1**;**  Node **\***T**=**Maxnum**(**t1**);**  T**->**l**=**t2**;**  **return** T**;**  **}**  void Split**(**int x**,**Node **\*&**t1**,**Node **\*&**t2**){**  Node **\***T**=**Find**(**x**);**  t1**=**T**->**l**;** t2**=**T**->**r**;**  **}**  void Inorder**(**Node **\***T**){**  **if** **(NULL==**T**)return** **;**  Inorder**(**T**->**l**);**  printf**(**"%d->"**,**T**->**key**);**  Inorder**(**T**->**r**);**  **}**  void \_Delete**(){**Delete**(**rt**);}**  void Delete**(**Node **\***T**){**  **if** **(NULL==**T**)return** **;**  Delete**(**T**->**l**);**  Delete**(**T**->**r**);**  free**(**T**);**  **}**  private**:**  Node **\***rt**;**//root  **};** |

AVL树

|  |
| --- |
| //codevs1285 莯ʕѸ˹  //by cww97  #include<cstdio>  #include<iostream>  #include<algorithm>  #define INF 0xfffffff  #define BASE 1000000  **using** **namespace** std**;**  int ans**=**0**;**  struct Node**{**  int x**,**bf**,**h**;**//bf=balance factor,h=height  Node **\***l**,\***r**;**  **};**  class AVLTree**{**  public**:**  void Init**()** **{** rt **=** **NULL;** **}**  int H**(**Node **\***T**){return** **(**T**==NULL)?**0**:**T**->**h**;}**  int BF**(**Node **\***l**,**Node **\***r**){**//get balance factor  **if** **(NULL==**l **&&** **NULL==**r**)** **return** 0**;**  **else** **if** **(NULL** **==** l**)** **return** **-**r**->**h**;**  **else** **if** **(NULL** **==** r**)** **return** l**->**h**;**  **return** l**->**h **-** r**->**h**;**  **}**  Node **\***Lrorate**(**Node **\***a**){**//left rorate  Node **\***b**;**  b**=**a**->**r**;**  a**->**r**=**b**->**l**;**  b**->**l**=**a**;**  a**->**h**=**max**(**H**(**a**->**l**),**H**(**a**->**r**))** **+** 1**;**  b**->**h**=**max**(**H**(**b**->**l**),**H**(**b**->**r**))** **+** 1**;**  a**->**bf**=**BF**(**a**->**l**,**a**->**r**);**  b**->**bf**=**BF**(**b**->**l**,**b**->**r**);**  **return** b**;**  **}**  Node **\***Rrorate**(**Node **\***a**){**//right rorate  Node **\***b**;**  b**=**a**->**l**;**  a**->**l**=**b**->**r**;**  b**->**r**=**a**;**  a**->**h**=**max**(**H**(**a**->**l**),**H**(**a**->**r**))** **+** 1**;**  b**->**h**=**max**(**H**(**b**->**l**),**H**(**b**->**r**))** **+** 1**;**  a**->**bf**=**BF**(**a**->**l**,**a**->**r**);**  b**->**bf**=**BF**(**b**->**l**,**b**->**r**);**  **return** b**;**  **}**  Node **\***LRrorate**(**Node **\***a**){**//left then right  a**->**l **=** Lrorate**(**a**->**l**);**  Node **\***c**;**  c**=**Rrorate**(**a**);**  **return** c**;**  **}**  Node **\***RLrorate**(**Node **\***a**){**//right then left  a**->**r**=**Rrorate**(**a**->**r**);**  Node **\***c**;**  c**=**Lrorate**(**a**);**  **return** c**;**  **}**  void Insert**(**int x**){**\_Insert**(**rt**,**x**);}**  void \_Insert **(**Node **\*&**T**,**int x**){**  **if** **(NULL==**T**){**  T**=(**Node**\*)**malloc**(sizeof(**Node**));**  T**->**x**=**x**;**  T**->**bf**=**0**;**T**->**h**=**1**;**  T**->**l**=**T**->**r**=NULL;**  **return** **;**  **}**  **if** **(**x **<** T**->**x**)** \_Insert**(**T**->**l**,**x**);**  **else** **if** **(**x **>** T**->**x**)** \_Insert**(**T**->**r**,**x**);**  **else** **return** **;** //error :the same y  T**->**h**=**max**(**H**(**T**->**l**),**H**(**T**->**r**))+**1**;**//maintain  T**->**bf**=**BF**(**T**->**l**,**T**->**r**);**  **if** **(**T**->**bf **>** 1 **||** T**->**bf **<** **-**1**){**//not balanced  **if** **(**T**->**bf **>** 0 **&&** T**->**l**->**bf **>** 0**)**T**=**Rrorate**(**T**);**  **else** **if** **(**T**->**bf **<** 0 **&&** T**->**r**->**bf **<** 0**)**T**=**Lrorate**(**T**);**  **else** **if** **(**T**->**bf **>** 0 **&&** T**->**l**->**bf **<** 0**)**T**=**LRrorate**(**T**);**  **else** **if** **(**T**->**bf **<** 0 **&&** T**->**r**->**bf **>** 0**)**T**=**RLrorate**(**T**);**  **}**  **}**  void GetPet**(**int x**){**//get pet or person  **if** **(NULL==**rt**){return** **;}**  int small**=**0**,**large**=**INF**;**  //printf("x=%d\n",x);  int flag**;**  **if** **(**Find**(**rt**,**x**,**small**,**large**)){**  printf**(**"find %d\n"**,**x**);**  \_Delete**(**rt**,**x**);**  **}else** **if** **(**small**==**0**)**flag**=**1**;**  **else** **if** **(**large**==**INF**)**flag**=**0**;**  **else** **if** **(**large**-**x**<**x**-**small**)**flag**=**1**;**  **else** flag**=**0**;**  **if** **(!**flag**){**//choose large  \_Delete**(**rt**,**small**);**  ans**=(**ans**+**x**-**small**)%**BASE**;**  **}else** **{**  \_Delete**(**rt**,**large**);**  ans**=(**ans**+**large**-**x**)%**BASE**;**  **}**  **}**  bool Find**(**Node **\***T**,**int x**,**int **&**small**,**int **&**large**){**  **if** **(NULL==**T**)return** 0**;**  **if** **(**x**==**T**->**x**)return** 1**;**  **if** **(**x**<**T**->**x**){**  large**=**min**(**large**,**T**->**x**);**  **return** Find**(**T**->**l**,**x**,**small**,**large**);**  **}else{**  small**=**max**(**small**,**T**->**x**);**  **return** Find**(**T**->**r**,**x**,**small**,**large**);**  **}**  **}**  void \_Delete**(**Node **\*&**T**,**int x**){**  **if** **(NULL==**T**)return** **;**  **if** **(**x **<** T**->**x**){**//y at left  \_Delete**(**T**->**l**,**x**);**  T**->**bf**=**BF**(**T**->**l**,**T**->**r**);**  **if** **(**T**->**bf**<-**1**){**  **if** **(**1**==**T**->**r**->**bf**)**T**=**RLrorate**(**T**);**  **else** T**=**Lrorate**(**T**);**//bf==0 or -1  **}**  **}else** **if** **(**x **>** T**->**x**){**//y at right  \_Delete**(**T**->**r**,**x**);**  T**->**bf**=**BF**(**T**->**l**,**T**->**r**);**  **if** **(**T**->**bf**>**1**){**  **if** **(-**1**==**T**->**l**->**bf**)**T**=**LRrorate**(**T**);**  **else** T**=**Rrorate**(**T**);**//bf==0 or 1  **}**  **}else** **{**//here is x  **if** **(**T**->**l**&&**T**->**r**){**//left &&right  Node **\***t**=**T**->**l**;**  **while** **(**t**->**r**)**t**=**t**->**r**;**  T**->**x**=**t**->**x**;**  \_Delete**(**T**->**l**,**t**->**x**);**  T**->**bf**=**BF**(**T**->**l**,**T**->**r**);**  **if** **(**T**->**bf**<-**1**){**  **if** **(**1**==**T**->**r**->**bf**)**T**=**RLrorate**(**T**);**  **else** T**=**Lrorate**(**T**);**//bf==0 or -1  **}**  **}else** **{**//left || right  Node **\***t**=**T**;**  **if** **(**T**->**l**)**T**=**T**->**l**;**  **else** **if(**T**->**r**)**T**=**T**->**r**;**  **else** **{**free**(**T**);**T**=NULL;}**  **if** **(**T**)**free**(**t**);**  **}**  **}**  **}**  //Debug,you will not need it at this problem  void show**(){**InOrder**(**rt**);**puts**(**"EndShow"**);}**  void InOrder**(**Node **\***T**){**//print l rt r  **if** **(NULL==**T**)return** **;**  InOrder**(**T**->**l**);**  printf**(**"%d "**,**T**->**x**);**  InOrder**(**T**->**r**);**  **}**  void Free**(){**FreeTree**(**rt**);}**  void FreeTree**(**Node **\***T**){**  **if** **(NULL==**T**)return** **;**  FreeTree**(**T**->**l**);**  FreeTree**(**T**->**r**);**  free**(**T**);**  **}**  private**:**  Node **\***rt**;**//root  **};**  int main**(){**  freopen**(**"fuck.in"**,**"r"**,**stdin**);**  int n**,**x**,**op**,**a**=**0**,**b**=**0**;**  scanf**(**"%d"**,&**n**);**  AVLTree T**;** T**.**Init**();**  **for** **(;**n**--;){**  scanf**(**"%d%d"**,&**op**,&**x**);**  //if pets>people put pets into the tree  //else put people into the tree  **if** **(**op**==**0**){**//come a pet  a**++;**  **if** **(**a**>**b**)**T**.**Insert**(**x**);**//more pet  **else** T**.**GetPet**(**x**);**//more people  **}else{**//come a person  b**++;**  **if** **(**a**<**b**)**T**.**Insert**(**x**);**//more people  **else** T**.**GetPet**(**x**);**//more pet  **}**  **}**  printf**(**"%d\n"**,**ans**%**BASE**);**  T**.**Free**();**  **return** 0**;**  **}** |

最小生成树（prim），hdu1102

|  |
| --- |
| #include<cstdio>  #include<cstring>  #include<iostream>  **using** **namespace** std**;**  const int N**=**107**;**  int n**,**g**[**N**][**N**];**  int prim**(){**  int minw**[**N**];**//MinWeight  bool used**[**N**];**  memset**(**used**,**0**,sizeof(**used**));**  memset**(**minw**,**0x7f**,sizeof(**minw**));**  minw**[**1**]=**0**;**  int sum**=**0**;**  **while** **(**1**){**  int v**=-**1**;**  **for** **(**int i**=**1**;**i**<=**n**;**i**++){**  **if** **(!**used**[**i**]&&(**v**==-**1**||**minw**[**i**]<**minw**[**v**]))**v**=**i**;**  **}**  **if** **(**v**==-**1**)break;**  used**[**v**]=**1**;**  sum**+=**minw**[**v**];**  **for** **(**int i**=**0**;**i**<=**n**;**i**++){**  minw**[**i**]=**min**(**minw**[**i**],**g**[**v**][**i**]);**  **}**  **}**  **return** sum**;**  **}**  int main**(){**  **for** **(;**scanf**(**"%d"**,&**n**)==**1**;){**  **for** **(**int i**=**1**;**i**<=**n**;**i**++)**  **for** **(**int j**=**1**;**j**<=**n**;**j**++)** scanf**(**"%d"**,&**g**[**i**][**j**]);**  int x**,**y**,**q**;**  scanf**(**"%d"**,&**q**);**  **for** **(;**q**--;){**  scanf**(**"%d%d"**,&**x**,&**y**);**  g**[**x**][**y**]=**g**[**y**][**x**]=**0**;**  **}**  printf**(**"%d\n"**,**prim**());**  **}**  **return** 0**;**  **}** |

次小生成树hdu4081

|  |
| --- |
| //hdu4081 次小生成树  #include<cmath>  #include<cstdio>  #include<cstring>  #include<iostream>  **using** **namespace** std**;**  const int N**=**1007**;**  int n**;**  double g**[**N**][**N**],**maxw**[**N**][**N**];**  bool used**[**N**][**N**];**  struct City**{**  int x**,**y**,**w**;**//w=population  City**():**x**(**0**),**y**(**0**),**w**(**0**){}**  City**(**int \_x**,**int \_y**,**int \_w**):**x**(**\_x**),**y**(**\_y**),**w**(**\_w**){}**  **}**citys**[**N**];**  double dist**(**City a**,**City b**){**  **return** sqrt**((**a**.**x**-**b**.**x**)\*(**a**.**x**-**b**.**x**)+(**a**.**y**-**b**.**y**)\*(**a**.**y**-**b**.**y**));**  **}**  double prim**(){**  int from**[**N**];**  bool vis**[**N**];**  double minw**[**N**];**  **for(**int i**=**1**;**i**<=**n**;**i**++){**  minw**[**i**]=**g**[**1**][**i**];**  from**[**i**]=**1**;**  **}**  memset**(**used**,**0**,sizeof(**used**));**  memset**(**maxw**,**0**,sizeof(**maxw**));**  memset**(**vis**,**0**,sizeof(**vis**));**  vis**[**1**]=**1**;**  minw**[**1**]=**0**;**  double sum**=**0**;**  **while** **(**1**){**  int v**=-**1**;**  **for** **(**int i**=**1**;**i**<=**n**;**i**++)if** **(!**vis**[**i**]){**  **if** **(**v**==-**1**||**minw**[**i**]<**minw**[**v**])**v**=**i**;**  **}**  **if** **(**v**==-**1**)break;**  used**[**v**][**from**[**v**]]=**used**[**from**[**v**]][**v**]=**1**;**  vis**[**v**]=**1**;**  sum**+=**minw**[**v**];**  **for** **(**int i**=**1**;**i**<=**n**;**i**++){**  **if** **(!**vis**[**i**]&&**g**[**v**][**i**]<**minw**[**i**]){**  minw**[**i**]=**g**[**v**][**i**];**  from**[**i**]=**v**;**  **}**  **if** **(**vis**[**i**]&&**i**!=**v**){**  maxw**[**i**][**v**]=**maxw**[**v**][**i**]=**max**(**maxw**[**i**][**from**[**v**]],**minw**[**v**]);**  **}**  **}**  **}**  **return** sum**;**  **}**  int main**(){**  //freopen("fuck.in","r",stdin);  int T**;**scanf**(**"%d"**,&**T**);**  **for** **(;**T**--;){**  scanf**(**"%d"**,&**n**);**int x**,**y**,**z**;**  **for** **(**int i**=**1**;**i**<=**n**;**i**++){**  scanf**(**"%d%d%d"**,&**x**,&**y**,&**z**);**  citys**[**i**]=**City**(**x**,**y**,**z**);**  **}**  **for** **(**int i**=**1**;**i**<=**n**;**i**++)for** **(**int j**=**1**;**j**<=**n**;**j**++)**  g**[**i**][**j**]=**dist**(**citys**[**i**],**citys**[**j**]);**  double sum**=**prim**(),**ans**=-**1**;**  **for** **(**int i**=**1**;**i**<=**n**;**i**++){**  **for** **(**int j**=**i**+**1**;**j**<=**n**;**j**++){**  **if** **(**used**[**i**][**j**])**ans**=**max**(**ans**,(**citys**[**i**].**w**+**citys**[**j**].**w**)/(**sum**-**g**[**i**][**j**]));**  **else** ans**=**max**(**ans**,(**citys**[**i**].**w**+**citys**[**j**].**w**)/(**sum**-**maxw**[**i**][**j**]));**  **}**  **}**  printf**(**"%.2lf\n"**,**ans**);**  **}**  **return** 0**;**  **}** |

最大流（Dinic）

|  |
| --- |
| #include<queue>  #include<cstdio>  #include<vector>  #include<string>  #include<cstring>  #include<iostream>  **using** **namespace** std**;**  **typedef** long long LL**;**  const int INF**=**0x3f3f3f3f**;**  const int N **=** 23 **<<** 1 **;**  int n**,**m**,**a**[**N**],**b**[**N**],**ans**[**N**][**N**];**  struct gragh**{**  struct Edge**{**  int from**,**to**,**cap**,**flow**;**  Edge**(**int u**,**int v**,**int c**,**int f**):**from**(**u**),**to**(**v**),**cap**(**c**),**flow**(**f**){}**  **};**  int s**,**t**;** //0=s,1~nАn+1~n+mPn+m+1=t  vector**<**Edge**>**edges**;**  vector**<** int**>**G**[**N**];** //gragh  bool vis**[**N**];** //use when bfs  int d**[**N**],**cur**[**N**];**//dist,now edge,use in dfs  inline void AddEdge**(**int from**,**int to**,**int cap**){**  edges**.**push\_back**(**Edge**(**from**,**to**,**cap**,**0**));**  edges**.**push\_back**(**Edge**(**to**,**from**,** 0 **,**0**));**  int top**=**edges**.**size**();**  G**[**from**].**push\_back**(**top**-**2**);**  G**[** to **].**push\_back**(**top**-**1**);**  **}**  inline bool BFS**(){**  memset**(**vis**,**0**,sizeof(**vis**));**  queue**<**int**>**Q**;**  Q**.**push**(**s**);**d**[**s**]=**0**;**vis**[**s**]=**1**;**  **while** **(!**Q**.**empty**()){**  int x**=**Q**.**front**();**Q**.**pop**();**  **for** **(**int i**=**0**;**i**<**G**[**x**].**size**();**i**++){**  Edge **&**e**=**edges**[**G**[**x**][**i**]];**  **if** **(**vis**[**e**.**to**]||**e**.**cap**<=**e**.**flow**)continue;**  vis**[**e**.**to**]=**1**;**  d**[**e**.**to**]=**d**[**x**]+**1**;**  Q**.**push**(**e**.**to**);**  **}**  **}**  **return** vis**[**t**];**  **}**  inline int DFS**(**const int**&** x**,**int a**){**  //printf("dfs:%d,%d\n",x,a);  **if** **(**x**==**t**||**a**==**0**){return** a**;}**  int flow **=** 0**,** f**;**  **for** **(**int**&** i**=**cur**[**x**];**i**<**G**[**x**].**size**();**i**++){**  Edge**&** e**=**edges**[**G**[**x**][**i**]];**  **if** **(**d**[**x**]+**1**!=**d**[**e**.**to**])continue;**  **if** **((**f**=**DFS**(**e**.**to**,**min**(**a**,**e**.**cap**-**e**.**flow**)))<=**0**)continue;**  e**.**flow **+=** f**;**  edges**[**G**[**x**][**i**]^**1**].**flow**-=**f**;**//ϲߍ  flow**+=**f**;** a**-=**f**;**  **if** **(**a**==**0**)** **break;**  **}**  **return** flow**;**  **}**  inline int maxflow**(){return** Maxflow**(**s**,**t**);}**  inline int Maxflow**(**const int**&** s**,**const int**&** t**){**  int flow**=**0**;**  memset**(**ans**,**0**,sizeof(**ans**));**  **while(**BFS**()){**  memset**(**cur**,**0**,sizeof(**cur**));**  int f **=** DFS**(**s**,**INF**);**  flow **+=** f **;**  **}**  **for** **(**int i**=**0**;**i**<**edges**.**size**();**i**++){**  Edge e**=**edges**[**i**];**  ans**[**e**.**from**][**e**.**to**-**n**]+=**e**.**flow**;**  **}**  **return** flow**;**  **}**  inline void Init**(){**  s **=** 0**,** t **=** n**+**m**+**1**;**  edges**.**clear**();**  **for** **(**int i**=**0**;**i**<=**m**+**n**+**1**;**i**++)** G**[**i**].**clear**()** **;**  **for** **(**int i**=**1**;**i**<=**n**;**i**++)** AddEdge**(** s **,**i**,**a**[**i**])** **;**  **for** **(**int i**=**1**;**i**<=**m**;**i**++)** AddEdge**(**i**+**n**,**t**,**b**[**i**])** **;**  **for** **(**int i**=**1**;**i**<=**n**;**i**++)**  **for** **(**int j**=**1**;**j**<=**m**;**j**++)**AddEdge**(**i**,**j**+**n**,**19**);**  **}**  **}**g**;** |

Truck 最大生成树+LCA

|  |
| --- |
| #include<cstdio>  #include<vector>  #include<cstring>  #include<iostream>  #include<algorithm>  **using** **namespace** std**;**  const int INF**=**0x3f3f3f3f**;**  const int N **=** 1e5 **+** 5**;**  int n**,**m**;**  struct gragh**{**  struct Edge**{**  int from**,**to**,**w**;**  Edge**(){}**  Edge**(**int x**,**int y**,**int z**):**from**(**x**),**to**(**y**),**w**(**z**){}**  bool **operator** **<** **(**const Edge**&** a**)**const**{**  **return** w **<** a**.**w**;**  **}**  **}**edges**[**N**],**be**[**N**];**  int E**,**f**[**N**],**fa**[**N**][**20**],**di**[**N**][**20**],**dep**[**N**];**  bool vis**[**N**];**  vector**<**int **>**G**[**N**];**  int F**(**int x**){**//鼯  **return** f**[**x**]==**x**?**x**:(**f**[**x**]=**F**(**f**[**x**]));**  **}**  inline void link**(**int x**,**int y**,**int z**){**  edges**[++**E**]=**Edge**(**x**,**y**,**z**);**  G**[**x**].**push\_back**(**E**);**  **}**  void build**(){**  E**=**0**;**  **for** **(**int i**=**1**;**i**<=**n**;**i**++)**G**[**i**].**clear**();**  int x**,**y**,**z**;**  **for** **(**int i**=**1**;**i**<=**n**;**i**++)**f**[**i**]=**i**;**  **for** **(**int i**=**1**;**i**<=**m**;**i**++){**  scanf**(**"%d%d%d"**,&**x**,&**y**,&**z**);**  be**[**i**]=**Edge**(**x**,**y**,**z**);**  f**[**F**(**x**)]=**F**(**y**);**  **}**  **}**  void kruskal**(){**  int treenum **=** 0**;**//forests  memset**(**vis**,**0**,sizeof(**vis**));**  **for** **(**int i**=**1**;**i**<=**n**;**i**++)if** **(!**vis**[**F**(**i**)]){**  treenum**++;**vis**[**F**(**i**)]=**1**;**  **}**  **for** **(**int i**=**1**;**i**<=**n**;**i**++)**f**[**i**]=**i**;**  sort**(**be**+**1**,**be**+**m**+**1**);**  int cnt **=** 0**;**  **for** **(**int i**=**m**;**i**>=**1**;**i**--){**  int x **=** be**[**i**].**from**;**  int y **=** be**[**i**].**to **;**  **if** **(**F**(**x**)==**F**(**y**))continue;**  f**[**F**(**x**)]=**F**(**y**);**  cnt**++;**  link**(**x**,**y**,**be**[**i**].**w**);**  link**(**y**,**x**,**be**[**i**].**w**);**  **if** **(**cnt**==**n**-**treenum**)break;**  **}**  **}**  void dfs**(**int x**){**  vis**[**x**]** **=** 1**;**  **for** **(**int i**=**1**;**i**<=**17**;**i**++){**  **if(**dep**[**x**]<(**1**<<**i**))break;**  fa**[**x**][**i**]=**fa**[**fa**[**x**][**i**-**1**]][**i**-**1**];**  di**[**x**][**i**]=**min**(**di**[**x**][**i**-**1**],**di**[**fa**[**x**][**i**-**1**]][**i**-**1**]);**  **}**  **for** **(**int i**=**0**;**i**<**G**[**x**].**size**();**i**++){**  Edge e **=** edges**[**G**[**x**][**i**]];**  **if** **(**vis**[**e**.**to**])continue;**  fa**[**e**.**to**][**0**]** **=** x**;**  di**[**e**.**to**][**0**]** **=** e**.**w**;**  dep**[**e**.**to**]** **=** dep**[**x**]+**1**;**  dfs**(**e**.**to**);**  **}**  **}**  int lca**(**int x**,**int y**){**  **if** **(**dep**[**x**]<**dep**[**y**])**swap**(**x**,**y**);**  int t **=** dep**[**x**]** **-** dep**[**y**];**  **for** **(**int i**=**0**;**i**<=**17**;**i**++)**  **if** **((**1**<<**i**)&**t**)** x **=** fa**[**x**][**i**];**  **for** **(**int i**=**17**;**i**>=**0**;**i**--)**  **if** **(**fa**[**x**][**i**]!=**fa**[**y**][**i**]){**  x**=**fa**[**x**][**i**];**y**=**fa**[**y**][**i**];**  **}**  **if** **(**x**==**y**)return** x**;**  **return** fa**[**x**][**0**];**  **}**  int ask**(**int x**,**int f**){**//f:father  int ans **=** INF**;**  int t **=** dep**[**x**]-**dep**[**f**];**  **for** **(**int i**=**0**;**i**<=**17**;**i**++)if(**t**&(**1**<<**i**)){**  ans**=**min**(**ans**,**di**[**x**][**i**]);**  x **=** fa**[**x**][**i**];**  **}**  **return** ans**;**  **}**  void work**(){**  build**();**  kruskal**();**  memset**(**vis**,**0**,sizeof(**vis**));**  **for** **(**int i**=**1**;**i**<=**n**;**i**++)if(!**vis**[**i**])**dfs**(**i**);**  int q**,**x**,**y**;**  scanf**(**"%d"**,&**q**);**  **while** **(**q**--){**  scanf**(**"%d%d"**,&**x**,&**y**);**  **if** **(**F**(**x**)!=**F**(**y**))**puts**(**"-1"**);**  **else** **{**  int t **=** lca**(**x**,**y**);**  x **=** ask**(**x**,**t**);**  y **=** ask**(**y**,**t**);**  printf**(**"%d\n"**,**min**(**x**,**y**));**  **}**  **}**  **}**  **}**g**;**  int main**(){**  //freopen("truck.in","r",stdin);  //freopen("truck.out","w",stdout);  **for** **(;~**scanf**(**"%d%d"**,&**n**,&**m**);)**g**.**work**();**  **return** 0**;**  **}** |

背包

|  |
| --- |
| #include <cstdio>  #include <cstring>  #include <algorithm>  **using** **namespace** std**;**  const int N **=** 100007**;**  struct node **{**  int v**,**w**,**n**;**  node**(){}**  node**(**int x**,**int y**,**int z**){**  v**=**x**,**w**=**y**,**n**=**z**;**  **}**  **}**a**[**N**];**  int f**[**N**];**  int main**(){**  //freopen("fuck.in","r",stdin);  int cash**,**n**,**x**,**y**;**  **for** **(;~**scanf**(**"%d%d"**,&**cash**,&**n**);){**  int A **=** 0**;**  **for(**int i**=**1**;**i**<=**n**;**i**++){**  scanf**(**"%d%d"**,&**x**,&**y**);**  **for** **(**int t**=**0**;(**1**<<**t**)<**x**;**t**++){**  int tt**=**1**<<**t**;**  a**[++**A**]=**node**(**y**\***tt**,**y**\***tt**,**1**);**  x **-=** tt**;**  **}**  **if** **(**x**)**a**[++**A**]=**node**(**y**\***x**,**y**\***x**,**1**);**  **}**  memset**(**f**,**0**,sizeof(**f**));//01背包**  **for** **(**int i**=**1**;**i**<=**A**;**i**++)**  **for** **(**int j**=**cash**;**j**>=**a**[**i**].**v**;**j**--)**  f**[**j**]=**max**(**f**[**j**],**f**[**j**-**a**[**i**].**v**]+**a**[**i**].**w**);**  int ans **=** 0**;//get ans**  **for** **(**int i**=**0**;**i**<=**cash**;**i**++)** ans**=**max**(**ans**,**f**[**i**]);**  printf**(**"%d\n"**,**ans**);**  **}**  **return** 0**;**  **}** |

多重背包通用模板（单调队列）

|  |
| --- |
| int f**[**N**];**  int va**[**N**],** vb**[**N**];**//MAX\_V  void pack**(**int V**,**int v**,**int w**,**int n**){**  **if** **(**n**==**0**||**v**==**0**)** **return;**  **if** **(**n**==**1**){**//01背包  **for** **(**int i**=**V**;**i**>=**v**;--**i**)**  f**[**i**]=**max**(**f**[**i**],**f**[**i**-**v**]+**w**);**  **return;**  **}**  **if** **(**n**\***v**>=**V**-**v**+**1**){**//多重背包(n >= V / v)  **for** **(**int i**=**v**;**i**<=**V**;++**i**)**  f**[**i**]=**max**(**f**[**i**],**f**[**i**-**v**]+**w**);**  **return;**  **}**  **for** **(**int j **=** 0 **;** j **<** v **;** **++**j **){**  int **\***pb **=** va**,** **\***pe **=** va **-** 1**;**  int **\***qb **=** vb**,** **\***qe **=** vb **-** 1**;**  **for** **(**int k**=**j**,**i**=**0**;**k**<=**V**;**k**+=**v**,++**i**){**  **if** **(**pe**==**pb**+**n**){**  **if(\***pb **==** **\***qb**)** **++**qb**;**  **++**pb**;**  **}**  int tt **=** f**[**k**]** **-** i **\*** w**;**  **\*++**pe **=** tt**;**  **while** **(**qe**>=**qb**&&** **\***qe**<**tt**)--**qe**;**  **\*++**qe **=** tt**;**  f**[**k**]** **=** **\***qb **+** i **\*** w**;**  **}**  **}**  **}**  //主程序调用  memset**(**f**,**0**,sizeof(**f**));** //pack  **for** **(**int i**=**1**;**i**<=**n**;**i**++)**  pack**(**cash**,**a**[**i**].**v**,**a**[**i**].**w**,**a**[**i**].**n**);**  int ans **=** 0**;** //getAns  **for** **(**int i**=**0**;**i**<=**cash**;**i**++)** ans**=**max**(**ans**,**f**[**i**]);**  printf**(**"%d\n"**,**ans**);** |