**Standard Code Library**

目录

[数据结构 - 5 -](#_Toc383281283)

[线段树 - 5 -](#_Toc383281284)

[函数式线段树 - 6 -](#_Toc383281285)

[Treap - 7 -](#_Toc383281286)

[序列维护Treap - 9 -](#_Toc383281287)

[持久化Treap - 12 -](#_Toc383281288)

[Splay - 13 -](#_Toc383281289)

[左偏树 - 15 -](#_Toc383281290)

[树状数组进阶使用 - 16 -](#_Toc383281291)

[树链剖分 - 16 -](#_Toc383281292)

[图论 - 17 -](#_Toc383281293)

[哈密顿回路/欧拉回路 - 17 -](#_Toc383281294)

[tarjan(directed) - 18 -](#_Toc383281295)

[Tanjan(undirected) （需更新 - 18 -](#_Toc383281296)

[2-SAT相关（需更新 - 20 -](#_Toc383281297)

[生成树相关 - 21 -](#_Toc383281298)

[最小树形图（需更新 - 21 -](#_Toc383281299)

[Bellman-Ford判负圈/差分约束系统（需更新 - 23 -](#_Toc383281300)

[Astar求K短路（需更新 - 23 -](#_Toc383281301)

[全局最小割（需更新 - 24 -](#_Toc383281302)

[倍增LCA - 25 -](#_Toc383281303)

[Hopcroft-Karp - 25 -](#_Toc383281304)

[Dinic - 26 -](#_Toc383281305)

[ISAP - 27 -](#_Toc383281306)

[费用流（bellman\_ford式） - 28 -](#_Toc383281307)

[费用流（zkw式）（需更新 - 29 -](#_Toc383281308)

[网络流建模Tips - 30 -](#_Toc383281309)

[一般图匹配 - 31 -](#_Toc383281310)

[Tuttx矩阵求一般图匹配数 - 32 -](#_Toc383281311)

[二分图最优匹配KM（需更新 - 33 -](#_Toc383281312)

[最大团最大独立集等知识 - 34 -](#_Toc383281313)

[数学 - 34 -](#_Toc383281314)

[分数类 - 34 -](#_Toc383281315)

[矩阵类 - 35 -](#_Toc383281316)

[FFT - 35 -](#_Toc383281317)

[自适应Simpson数值积分 - 37 -](#_Toc383281318)

[几何 - 37 -](#_Toc383281319)

[2D（未完成 - 37 -](#_Toc383281320)

[字符串 - 40 -](#_Toc383281321)

[KMP - 40 -](#_Toc383281322)

[扩展kmp - 40 -](#_Toc383281323)

[Manacher - 41 -](#_Toc383281324)

[最小表示法 - 41 -](#_Toc383281325)

[AC自动机 - 41 -](#_Toc383281326)

[后缀数组 - 42 -](#_Toc383281327)

[杂物 - 43 -](#_Toc383281328)

[单调队列/栈的使用Tips - 43 -](#_Toc383281329)

[高斯消元 同余模方程/异或方程组 - 43 -](#_Toc383281330)

[多项式乘法替代矩阵求线性递推数列 - 44 -](#_Toc383281331)

[树的点分治 - 45 -](#_Toc383281332)

[斯坦纳树 - 47 -](#_Toc383281333)

[矩形面积并 - 49 -](#_Toc383281334)

## 数据结构

### 线段树

|  |
| --- |
| struct stree  {    long long sum , multi , add;  }t[N << 1];  int id(int l , int r) {return l + r | l != r;}  #define MID int mid = (l + r) >> 1;  #define ID id(l , r)  #define Left l , mid  #define Right mid + 1 , r  void pushup(int l , int r)  {    MID; int p = ID , LL = id(Left) , RR = id(Right);    t[p].sum = (t[LL].sum + t[RR].sum) % Q;  }  void pushdown(int l , int r)  {    MID; int p = ID , LL = id(Left) , RR = id(Right);    if (t[p].multi != 1)    {      t[LL].multi = t[LL].multi \* t[p].multi % Q;      t[LL].sum = t[LL].sum \* t[p].multi % Q;      t[LL].add = t[LL].add \* t[p].multi % Q;      t[RR].multi = t[RR].multi \* t[p].multi % Q;      t[RR].sum = t[RR].sum \* t[p].multi % Q;      t[RR].add = t[RR].add \* t[p].multi % Q;      t[p].multi = 1;    }    if (t[p].add)    {      t[LL].sum = (t[LL].sum + t[p].add \* (mid - l + 1)) % Q;      t[LL].add = (t[LL].add + t[p].add) % Q;      t[RR].sum = (t[RR].sum + t[p].add \* (r - mid)) % Q;      t[RR].add = (t[RR].add + t[p].add) % Q;      t[p].add = 0;    }  }  void update(int l , int r , int top , int bot , int q , int w)  {    if (top <= l && r <= bot) {int p = ID;      if (q == 2) t[p].add = (t[p].add + w) % Q , t[p].sum = (t[p].sum + w \* (r - l + 1)) % Q;      if (q == 1) t[p].multi = t[p].multi \* w % Q , t[p].sum = t[p].sum \* w % Q , t[p].add = t[p].add \* w % Q;      return;}MID;pushdown(l , r);    if (top <= mid) update(Left , top , bot , q , w);    if (bot > mid) update(Right , top , bot , q , w);    pushup(l , r);  }  int Query(int l , int r , int top , int bot)  {    if (top <= l && r <= bot) {return t[ID].sum;}MID;pushdown(l , r);int ans = 0;    if (top <= mid) ans += Query(Left , top , bot);    if (bot > mid) ans += Query(Right , top , bot);    pushup(l , r); return ans % Q;  }  void Build(int l , int r)  {    t[ID].multi = 1;    if (l == r) {t[ID].sum = a[l];return;}    MID;Build(Left);Build(Right);pushup(l , r);  } |

注：

(1)Pushdown下传标记的顺序

(2)两种线段树的标号方案，不用吝啬空间。

(3)可离散化对值域建线段树完成系列操作

(4)动态开线段树节点，上面板子都不用看了 = =

### 函数式线段树

|  |
| --- |
| #define N 10005  int n , m , a[N];  int D = 1e9;  int c[N];  #define Node 8388608  #define MID int mid = l + r >> 1  #define Left l , mid  #define Right mid + 1 , r  int ch[Node][2] , sum[Node] , nodecnt;  int newnode()  {  ++ nodecnt;  ch[nodecnt][0] = ch[nodecnt][1] = sum[nodecnt] = 0;  return nodecnt;  }  void add(int& p , int l , int r , int x , int w)  {  if (!p) p = newnode();  if (l == r)  sum[p] += w;  else {  MID;  if (x <= mid)  add(ch[p][0] , Left , x , w);  else  add(ch[p][1] , Right , x , w);  sum[p] = sum[ch[p][0]] + sum[ch[p][1]];  }  }  void add(int x , int w , int val)  {  for (int i = x ; i <= n ; i += i & -i)  add(c[i] , 0 , D , w , val);  }  int PP[50] , MM[50] , sp , sm;  int query(int l , int r , int K)  {  if (l == r) return l; MID;  int cnt = 0;  for (int i = 0 ; i < sp ; ++ i) cnt += sum[ch[PP[i]][0]];  for (int i = 0 ; i < sm ; ++ i) cnt -= sum[ch[MM[i]][0]];  if (cnt >= K) {  for (int i = 0 ; i < sp ; ++ i) PP[i] = ch[PP[i]][0];  for (int i = 0 ; i < sm ; ++ i) MM[i] = ch[MM[i]][0];  return query(Left , K);  } else {  for (int i = 0 ; i < sp ; ++ i) PP[i] = ch[PP[i]][1];  for (int i = 0 ; i < sm ; ++ i) MM[i] = ch[MM[i]][1];  return query(Right , K - cnt);  }  }  void work()  {  int i , j , k , x , y;  char str[5];  scanf("%d%d",&n,&m);  for (i = 1 ; i <= n ; ++ i)  scanf("%d",&a[i]) , add(i , a[i] , 1);  for (i = 1 ; i <= m ; ++ i) {  scanf("%s%d%d",str,&x,&y);  if (\*str == 'Q') {  scanf("%d",&k);  sp = 0 ; for (j = y ; j ; j -= j & -j) PP[sp ++] = c[j];  sm = 0 ; for (j = x-1 ; j ; j -= j & -j) MM[sm ++] = c[j];  printf("%d\n" , query(0 , D , k));  } else {  add(x , a[x] , -1);  a[x] = y;  add(x , a[x] , 1);  }  }  }  int main()  {  freopen("~input.txt" , "r" , stdin);  work();  return 0;  } |

以上是动态区间第K大的代码，树状数组套线段树，动态开节点。

然后就是比如保留历史版本啊，线段树套线段树（树上动态第K大）啊，相信现在已经不难了。

### Treap

|  |
| --- |
| struct Treap  {    int root , nodecnt , priority[N] , child[N][2];    int key[N];    int Size[N] , cnt[N];    vector<int> del;    Treap(){      del.clear();      root = 0 , nodecnt = 1;      priority[0] = key[0] = 1 << 30;      Size[0] = 0;    }    void clear()    {      del.clear();      root = 0 , nodecnt = 1;      priority[0] = -1 << 30 , key[0] = -1<<30;      Size[0] = 0;    }    int size()    {      return Size[root];    }    void update(int x) //更新函数，重要    {      Size[x] = Size[child[x][0]] + Size[child[x][1]] + cnt[x];    }    void rotate (int& x , int t)    {      int y = child[x][t];      child[x][t] = child[y][t ^ 1];      child[y][t ^ 1] = x;      update(x) , update(y);      x = y;    }    void \_\_insert(int& x , int k)    {      if (x)      {        if (key[x] == k)          ++ cnt[x];        else        {          int t = key[x] < k;          \_\_insert(child[x][t] , k);          if (priority[child[x][t]] > priority[x])             rotate(x , t);        }      }      else      {        if (del.size())//视情况看是否需要垃圾回收        {          x = del.back();          del.pop\_back();        }        else x = nodecnt ++;        key[x] = k;        cnt[x] = 1;        priority[x] = rand();        child[x][0] = child[x][1] = 0;      }      update(x);    }    void query(int x , int l , int r , int& ans)    {//询问关键字在一定范围的另一关键字的权重，小心常数别写错。      if(!x) return; int val = key[x].first;      if( l <= val && val <=r)      {        ans = \_\_gcd(ans, val);        if(l == -1 << 30)        {          ans = \_\_gcd(ans, G[child[x][0]]);          query(child[x][1] , l, r, ans);        }        else if(r == 1 << 30)        {          query(child[x][0] , l, r, ans);          ans = \_\_gcd(ans, G[child[x][1]]);        }        else        {          query(child[x][0] , l, 1 << 30 , ans);          query(child[x][1] , -1 << 30, r , ans);        }        return;      }      if(r < val)          query(child[x][0] , l, r, ans);      if(val < l)          query(child[x][1] , l, r, ans);    }    int lower\_bound(int x , int val , int opt)    {//很普通的lower\_bound，灵活改写      if(!x) return opt;      if (key[x].first >= val)        return lower\_bound(child[x][0] , val , key[x].first);      else        return lower\_bound(child[x][1], val , opt);    }    void \_\_erase(int& x , int k)    {      if (!x) return;      if (key[x] == k)      {        if (cnt[x] > 1)          -- cnt[x];        else        {          if (!child[x][0] && !child[x][1])          {            del.push\_back(x);            x = 0;            return;          }          int t = priority[child[x][0]] < priority[child[x][1]];          rotate(x , t);          \_\_erase(x , k);        }      }      else        \_\_erase(child[x][key[x] < k] , k);      update(x);    }    int \_\_Rank(int& x , int k)    {      if(!x) return 0;      if (k < key[x])        return \_\_Rank(child[x][0] , k);      int Rk = Size[child[x][0]] + cnt[x];      if (k > key[x])        Rk += \_\_Rank(child[x][1] , k);      return Rk;    }    int \_\_getKth(int& x , int k)    {      if (k <= Size[child[x][0]])        return \_\_getKth(child[x][0] , k);      k -= Size[child[x][0]] + cnt[x];      if (k <= 0)  return key[x];      return \_\_getKth(child[x][1] , k);    }      void insert(int x){      \_\_insert(root , x);}    void erase(int x){      \_\_erase(root , x);}    int \_\_Rank(int x){      return \_\_Rank(root , x);}    int getKth(int x){      return \_\_getKth(root , x);}  };  虽然上面的代码又臭又长……但还是挺好用的，调能用的用。 |

### 序列维护Treap

Splay的替代品，比splay好写的多，速度略慢？

|  |
| --- |
| #include <iostream>  #include <cstdio>  #include <cstdlib>  #include <cstring>  using namespace std;  typedef long long LL;  #define N 200505  struct Treap  {  int nodecnt;  int L[N] , R[N] , cnt[N];  int key[N];  int Min[N] , add[N] , rev[N];  void clear() {  nodecnt = 0;  }  Treap () {clear();}  bool hey(int A , int B) {  return (LL)rand() \* (cnt[A] + cnt[B]) < (LL)cnt[A] \* RAND\_MAX;  }  int newnode(int val) {  ++ nodecnt , L[nodecnt] = R[nodecnt] = 0;  cnt[nodecnt] = 1 , Min[nodecnt] = key[nodecnt] = val , rev[nodecnt] = add[nodecnt] = 0;  return nodecnt;  }  void pushup(int x) {  cnt[x] = 1 , Min[x] = key[x];  if (L[x]) cnt[x] += cnt[L[x]] , Min[x] = min(Min[x] , Min[L[x]]);  if (R[x]) cnt[x] += cnt[R[x]] , Min[x] = min(Min[x] , Min[R[x]]);  }  void pushdown(int x) {  if (rev[x]) {  if (L[x]) rev[L[x]] ^= 1 , swap(L[L[x]] , R[L[x]]);  if (R[x]) rev[R[x]] ^= 1 , swap(L[R[x]] , R[R[x]]);  rev[x] = 0;  }  if (add[x]) {  if (L[x]) add[L[x]] += add[x] , Min[L[x]] += add[x] , key[L[x]] += add[x];  if (R[x]) add[R[x]] += add[x] , Min[R[x]] += add[x] , key[R[x]] += add[x];;  add[x] = 0;  }  }  void merge(int& p , int x , int y) {  if (!x || !y)  p = x | y;  else if ( hey(x , y) ) // key[x] < key[y]  pushdown(x) , merge(R[x] , R[x] , y) , pushup(p = x);  else  pushdown(y) , merge(L[y] , x , L[y]) , pushup(p = y);  }  void split(int p , int& x , int& y , int size) {  if (!size) {  x = 0 , y = p;  return;  } pushdown(p);  if (cnt[L[p]] >= size)  y = p , split(L[p] , x , L[y] , size) , pushup(y);  else  x = p , split(R[p] , R[x] , y , size - cnt[L[p]] - 1) , pushup(x);  }  void Build(int \*A , int& p , int l , int r) {  int mid = (l + r) >> 1;  p = newnode(A[mid]);  if (l < mid) Build(A , L[p] , l , mid - 1);  if (mid < r) Build(A , R[p] , mid + 1 , r);  pushup(p);  }  };  Treap T;  int n , m , a[N];  void work()  {  int i , j , k , x , y , z;  char str[10];  int root = 0;  scanf("%d",&n);  for (i = 1 ; i <= n ; ++ i)  scanf("%d",&a[i]);  T.Build(a , root , 1 , n);  scanf("%d",&m);  while (m --) {  scanf("%s" , str);  if (\*str == 'A') {  scanf("%d%d%d",&j , &k , &z);  T.split(root , root , x , j - 1);  T.split(x , x , y , k - j + 1);  T.add[x] += z , T.key[x] += z , T.Min[x] += z;  T.merge(x , x , y);  T.merge(root , root , x);  }  if (\*str == 'I') {  scanf("%d%d",&j , &z);  x = T.newnode(z);  T.split(root , root , y , j);  T.merge(root , root , x);  T.merge(root , root , y);  }  if (\*str == 'D') {  scanf("%d",&j);  T.split(root , root , x , j - 1);  T.split(x , x , y , 1);  T.merge(root , root , y);  }  if (\*str == 'M') {  scanf("%d%d",&j , &k);  T.split(root , root , x , j - 1);  T.split(x , x , y , k - j + 1);  printf("%d\n" , T.Min[x]);  T.merge(x , x , y);  T.merge(root , root , x);  }  if (\*str == 'R' && \*(str + 3) == 'E')  {  scanf("%d%d",&j , &k);  T.split(root , root , x , j - 1);  T.split(x , x , y , k - j + 1);  T.rev[x] ^= 1 , swap(T.L[x] , T.R[x]);  T.merge(x , x , y);  T.merge(root , root , x);  }  if (\*str == 'R' && \*(str + 3) == 'O')  {  scanf("%d%d%d", &j , &k , &z);  z %= (k - j + 1); if (!z) continue;  T.split(root , root , x , j - 1);  T.split(x , x , y , k - j + 1);  T.split(x , j , k , k - j + 1 - z);  T.merge(x , k , j);  T.merge(x , x , y);  T.merge(root , root , x);  }  }  }  int main()  {  work();  return 0;  }  POJ3580  区间加区间翻转区间循环移动区间极值单点插入单点删除  代码中的query极值是从fhq博客所学。 |

### 持久化Treap

|  |
| --- |
| 泰国某赛区LRJ出的一道喜闻乐见的询问历史版本的文本处理器。  typedef long long LL;  #define N 5000005  int root[50005] , c;  struct Treap  {  int nodecnt;  int L[N] , R[N] , cnt[N];  char key[N];  void clear() {  nodecnt = 0;  }  Treap () {clear();}  bool hey(int A , int B) {  return (LL)rand() \* (cnt[A] + cnt[B]) < (LL)cnt[A] \* RAND\_MAX;  }  int newnode(char val) {  ++ nodecnt , L[nodecnt] = R[nodecnt] = 0;  cnt[nodecnt] = 1 , key[nodecnt] = val;  return nodecnt;  }  int copynode(int A) {  if (!A) return 0;  ++ nodecnt , L[nodecnt] = L[A] , R[nodecnt] = R[A];  cnt[nodecnt] = cnt[A] , key[nodecnt] = key[A];  return nodecnt;  }  void pushup(int x) {  cnt[x] = 1;  if (L[x]) cnt[x] += cnt[L[x]];  if (R[x]) cnt[x] += cnt[R[x]];  }  void merge(int& p , int x , int y) {  if (!x || !y) {  p = 0;  if (x) p = copynode(x);  if (y) p = copynode(y);  }  else if ( hey(x , y) ) {  p = copynode(x);  merge(R[p] , R[x] , y) , pushup(p);  }  else {  p = copynode(y);  merge(L[p] , x , L[y]) , pushup(p);  }  }  void split(int p , int& x , int& y , int size) {  if (!size) {  x = 0 , y = copynode(p);  return;  }  if (cnt[L[p]] >= size) {  y = copynode(p);  split(L[p] , x , L[y] , size) , pushup(y);  }  else {  x = copynode(p);  split(R[p] , R[x] , y , size - cnt[L[p]] - 1) , pushup(x);  }  }  void Build(char \*A , int& p , int l , int r) {  int mid = (l + r) >> 1;  p = newnode(A[mid]);  if (l < mid) Build(A , L[p] , l , mid - 1);  if (mid < r) Build(A , R[p] , mid + 1 , r);  pushup(p);  }  void print(int p) {  if (L[p]) print(L[p]);  putchar(key[p]); c += key[p] == 'c';  if (R[p]) print(R[p]);  }  };  Treap T;  int n , m;  char str[100005];  void work()  {  int i , j , k , x , y , z;  int version = 0;  scanf("%d",&n);  while (n --) {  scanf("%d",&i);  if (i == 1) {  scanf("%d%s" , &j , str) , j -= c;  T.Build(str , x , 0 , strlen(str) - 1);  T.split(root[version] , y , z , j);  T.merge(y , y , x);  T.merge(root[++ version] , y , z);  }  else if (i == 2) {  scanf("%d%d",&j,&k) , j -= c , k -= c;  T.split(root[version] , x , y , j - 1);  T.split(y , y , z , k);  T.merge(root[++ version] , x , z);  }  else {  scanf("%d%d%d",&i,&j,&k);  i -= c , j -= c , k -= c;  T.split(root[i] , x , y , j - 1);  T.split(y , y , z , k);  T.print(y);  puts("");  }  }  } |

### Splay

|  |
| --- |
| struct SplayTree  {     //没有pushdown的时候可以去掉stack简化代码      int nodecnt , root , type[N] , parent[N] , childs[N][2] , size[N] , st[N];      //addtional data      ULL key[N] , hash[N];        void clear()      {          root = 0;          size[root] = 0;          nodecnt = 1;      }        int malloc(ULL x)      {          type[nodecnt] = 2;          childs[nodecnt][0] = childs[nodecnt][1] = 0;          size[nodecnt] = 1;          key[nodecnt] = hash[nodecnt] = x;          return nodecnt ++;      }        void pushup(int x)      {          size[x] = size[childs[x][0]] + 1 + size[childs[x][1]];          hash[x] = (hash[childs[x][0]] \* MAGIC + key[x]) \* power[size[childs[x][1]]] + hash[childs[x][1]];      }      void pushdown(int x)      {        // 区间翻转时注意type的变化  //无非是交换儿子以及维护儿子的type      }      void rotate(int x)      {          int t = type[x] , y = parent[x] , z = childs[x][1 - t];          type[x] = type[y];          parent[x] = parent[y];          if (type[x] != 2)              childs[parent[x]][type[x]] = x;          type[y] = 1 - t;          parent[y] = x;          childs[x][1 - t] = y;          if (z)              type[z] = t , parent[z] = y;          childs[y][t] = z;          pushup(y);      }      void splay(int x)      {          int scnt = 0;          st[scnt ++] = x;          for (int i = x ; type[i] != 2 ; i = parent[i])              st[scnt ++] = parent[i];  for (int i = scnt - 1 ; i >= 0 ; -- i)              pushdown(st[i]);          while (type[x] != 2)          {              int y = parent[x];              if (type[y] == type[x])                  rotate(y);              else rotate(x);              if (type[x] != 2)                  rotate(x);          }          pushup(x);      }        int getKth(int x , int k)      {          while (x)          {              pushdown(x);              if (size[childs[x][0]] + 1 == k)                  break;              if (k <= size[childs[x][0]])                  x = childs[x][0];              else k -= size[childs[x][0]] + 1 , x = childs[x][1];          }          return x;      }      int Rank(int x)      {          splay(x) , root = x;          return size[childs[x][0]];      }      void split(int& x , int& y , int a)      {          if (a == size[x]) {y = 0;return;}          //if (a == 0) {y = x ; x = 0;return;}          y = getKth(x , a + 1);          splay(y);          x = childs[y][0];          type[x] = 2;          childs[y][0] = 0;          pushup(y);      }      void modify(int x , int k , ULL t)      {     if (k == size[childs[x][0]] + 1)         key[x] = hash[x] = t;     else if (k <= size[childs[x][0]])       modify(childs[x][0] , k , t);  else modify(childs[x][1] , k - size[childs[x][0]] - 1 , t);     pushup(x);  }      void merge(int& x , int& y)      {          if (!x || !y) {x = x | y ; return;}          x = getKth(x , size[x]);          splay(x);          childs[x][1] = y;          type[y] = 1;          parent[y] = x;          pushup(x);      }      int Build(int l , int r)      {          int mid = (l + r) >> 1;          int x = malloc(s[mid] - 'a' + 1);          if (l <= mid - 1) childs[x][0] = Build(l , mid - 1) , parent[childs[x][0]] = x , type[childs[x][0]] = 0;          if (mid + 1 <= r) childs[x][1] = Build(mid + 1 , r) , parent[childs[x][1]] = x , type[childs[x][1]] = 1;          pushup(x);          return x;      }  }; |

### 左偏树

|  |
| --- |
| int n , a[N] , d[N] , f[N] , m , l[N] , r[N];  int getf(int x) {return x == f[x] ? x : f[x] = getf(f[x]);}  int merge(int x , int y)  {  if (!x || !y) return x | y;  if (a[x] < a[y]) swap(x , y);  r[x] = merge(r[x] , y) , f[r[x]] = x;  if (d[l[x]] < d[r[x]])  swap(l[x] , r[x]);  d[x] = r[x] ? d[r[x]] + 1 : 0;  return x;  }  合并两个集合要有下面的f[i] = f[j] = k;  i = merge(l[xx] , r[xx]);  a[xx] >>= 1 , l[xx] = r[xx] = d[xx] = 0;  i = merge(i , xx);  j = merge(l[yy] , r[yy]);  a[yy] >>= 1 , l[yy] = r[yy] = d[yy] = 0;  j = merge(j , yy);  k = merge(i , j);  printf("%d\n" , a[k]);  f[i] = f[j] = k; |

### 树状数组进阶使用

|  |
| --- |
| 树状数组里二分  int **getKth**(int k)  {  int x = 0 , i;  **for** (i = 16 ; i >= 0 ; -- i)  **if** (x + (1 << i) <= D && c[x + (1 << i)] < k)  {  x += 1 << i;  k -= c[x];  }  **return** x + 1;  } |
| 中二树，区间加求和，维护一个线性函数Ax+B  struct CHU\_2\_BIT  {  int n;  LL B[N] , C[N];  void init(int size) {  n = size;  memset(B , 0 , sizeof(B));  memset(C , 0 , sizeof(C));  }  CHU\_2\_BIT() {}  CHU\_2\_BIT(int size) {  init(size);  }  void \_add(LL\* c , int x , LL w) {  for ( ; x <= n ; x += x & -x)  c[x] += w;  }  LL \_sum(LL\* c , int x) {  LL res = 0;  for ( ; x > 0 ; x -= x & -x)  res += c[x];  return res;  }  void add(int l , int r , LL w) {  \_add(B , l , w) , \_add(B , r + 1 , -w);  \_add(C , l , w \* l) , \_add(C , r + 1 , -w \* (r + 1));  }  LL sum(int l , int r) {  LL res = 0;  res += (r + 1) \* \_sum(B , r) - l \* \_sum(B , l - 1);  res -= \_sum(C , r) - \_sum(C , l - 1);  return res;  }  }T;  二维的加减求和相信自己也能手推出来…… |

### 树链剖分

|  |
| --- |
| int size[N] , heavy[N] , tid[N] , Top[N] , ncnt , dep[N] , Fa[N];  // size : 子树大小  // heavy : 子树大小最大的子树（重边  // tid : 结点由重边串起来时的编号，在数据结构中维护  // Top : 结点所在重链的头  // Dep : 结点深度  // Fa : 结点父亲  void FindHE(int x , int fa , int Dep) //找重边重儿子  {  heavy[x] = 0 , size[x] = 1 , dep[x] = Dep , Fa[x] = fa;  for (int i = pre[x] ; ~i ; i = e[i].next)  if (e[i].x != fa)  {  int y = e[i].x;  FindHE(y , x , Dep + 1);  size[x] += size[y];  if (!heavy[x] || size[y] > size[heavy[x]])  heavy[x] = y;  }  }  void ConnectHE(int x , int anc) // 把重边串起来形成重链  {  tid[x] = ++ ncnt , Top[x] = anc;  update(1 , n , ncnt , a[x]); //单点更新值，顺便  if (heavy[x])  ConnectHE(heavy[x] , anc);  for (int i = pre[x] ; ~i ; i = e[i].next)  if (e[i].x != Fa[x] && e[i].x != heavy[x])  ConnectHE(e[i].x , e[i].x);  }  int query(int x , int y) // 询问  {  int ans = -1 << 30;  while (Top[x] != Top[y]) // 两个结点不在一条重链上  {  if (dep[Top[x]] < dep[Top[y]])  swap(x , y); // 让深度大的那条链整个遍历完，tid[Top[x]]其中包含最头那条轻边  ans = max(ans , query(1 , n , tid[Top[x]] , tid[x]));  x = Fa[Top[x]];  }  if (dep[x] > dep[y]) swap(x , y);//在一条链上了  ans = max(ans , query(1 , n , tid[x] , tid[y]));  return ans;  } |

## 图论

### 哈密顿回路/欧拉回路

|  |
| --- |
| 欧拉回路的判定  大前提是联通  无向图所有点的度数为偶数  有向图所有点入度等于出度  欧拉通路的判定  大前提是联通  无向图除了两个点所有点的度数为偶数  有向图除了两个点所有点入度等于出度，两个点中一个入多1一个出多1  遍历（大概是这个样子/有向）  void dfs(int x , int z)  {  for (int i = 0 ; i < d[x] ; ++ i)  if (!v[e[x][i].second])  {  v[e[x][i].second] = 1;  dfs(e[x][i].first , e[x][i].second);  }  if (z != -1)  ans.pb(mp(z , x != a[z]? '+' : '-'));  }  无向的话只需把反向边也标记了就行  哈密顿回路满足条件有定理可以构造，也可以看看图的特殊性是否可以转化为欧拉回路 |

### tarjan(directed)

|  |
| --- |
| int ncnt , scnt , bel[N] , low[N] , dfn[N];  int f[N];  stack<int> S;  void dfs(int x) {  int i , y;  low[x] = dfn[x] = ++ ncnt;  f[x] = 1 , S.push(x);  for (i = pre[x] ; ~i ; i = e[i].next) {  y = e[i].x;  if (!dfn[y]) {  dfs(y);  low[x] = min(low[x] , low[y]);  } else if (f[y])  low[x] = min(low[x] , dfn[y]);  }  if (low[x] == dfn[x]) {  val[scnt] = 0;  do {  i = S.top() , S.pop() , f[i] = 0;  bel[i] = scnt , val[scnt] += v[i];  }while (i != x);  ++ scnt;  }  } |

### Tanjan(undirected) （需更新

|  |
| --- |
| int n , DFN[N] , low[N] , index , bcnt , bel[N];  bool iscut[N];  vector<int> e[N];  vector<int> bcc[N];  stack< pair<int,int> > S;  void dfs(int x , int fa)  {    low[x] = DFN[x] = ++ index;    int i , y , xx , yy , child = 0;    for (i = 0 ;i < (int)e[x].size() ; ++ i)    {      y = e[x][i];      if (!DFN[y])      {        S.push(mp(x,y));        ++ child;        dfs(y , i ^ 1);        low[x] = min(low[x] , low[y]);        //if (low[y] > DFN[x]) (x,y) is bridge        if (low[y] >= DFN[x])        {          iscut[x] = 1;          ++ bcnt , bcc[bcnt].clear();          do          {            xx = S.top().fi , yy = S.top().se , S.pop();            if (bel[xx] != bcnt) bcc[bcnt].pb(xx) , bel[xx] = bcnt;            if (bel[yy] != bcnt) bcc[bcnt].pb(yy) , bel[yy] = bcnt;          }while(x != xx || y != yy);        }      }      else if (DFN[y] < DFN[x] && i != fa)        S.push(mp(x,y)) , low[x] = min(DFN[y] , low[x]);    }    if (fa < 0 && child == 1) iscut[x] = 0;  }  **附：一份完整的点双联通缩点为树代码**  **询问两条边之间要路过多少个割点**  **int** n **,** m **,** pre**[**N**] ,** mcnt **,** id**[**N**];**  struct edge  {  int x , next;  }e[M] , g[M << 1];  int pp[N] , cc;  **int** low**[**N**] ,** DFN**[**N**] ,** ncnt **,** bel**[**M**] ,** bcnt **,** cutnum **,** pel**[**M**];**  vector**<int>** bcc**[**N**];**  **bool** iscut**[**N**];**  stack< pair<int,int> > S;  void tarjan(int x , int fa)  {  low**[**x**] =** DFN**[**x**] = ++** ncnt**;**  **int** i **,** y **,** xx **,** yy **,** child **=** 0**;**  for (i = pre[x] ; ~i ; i = e[i].next)  {  y **=** e**[**i**].**x**;**  if (!DFN[y])  {  S**.**push**(**mp**(**x **,** i**));**  ++ child;  tarjan**(**y **,** i**);**  low**[**x**] =** min**(**low**[**x**] ,** low**[**y**]);**  if (low[y] >= DFN[x])  {  iscut**[**x**] =** 1**;**  ++ bcnt , bcc[bcnt].clear();  do  {  xx **=** S**.**top**().**fi **,** yy **=** e**[**S**.**top**().**se**].**x **,** pel**[(**S**.**top**().**se **>>** 1**) +** 1**] =** bcnt **,** S**.**pop**();**  if (bel[xx] != bcnt) bcc[bcnt].pb(xx) , bel[xx] = bcnt;  if (bel[yy] != bcnt) bcc[bcnt].pb(yy) , bel[yy] = bcnt;  }while(x != xx || y != yy);  }  }  else if (DFN[y] < DFN[x] && ((i ^ 1) != fa))  S**.**push**(**mp**(**x **,** i**)) ,** low**[**x**] =** min**(**DFN**[**y**] ,** low**[**x**]);**  }  if (fa < 0 && child == 1) iscut[x] = 0;  }  int f[N][20] , L[N];  bool vis[N];  void dfs(int x , int fa)  {  f[x][0] = fa , L[x] = L[fa] + 1 , vis[x] = 1;  for (int i = pp[x] ; ~i ; i = g[i].next)  if (!vis[g[i].x])  dfs**(**g**[**i**].**x **,** x**);**  }  int LCA(int x , int y)  {  int log , i; if (x == y) return x;  if (L[x] < L[y]) swap(x , y);  for (log = 1 ; (1 << log) <= L[x] ; ++ log); -- log;  for (i = log ; i >= 0 ; -- i)  if (L[x] - (1 << i) >= L[y])  x **=** f**[**x**][**i**];**  if (x == y) return y;  for (i = log ; i >= 0 ; -- i)  if (f[x][i] && f[y][i] && f[x][i] != f[y][i])  x **=** f**[**x**][**i**] ,** y **=** f**[**y**][**i**];**  return f[x][0];  }  void work()  {  int i , j , x , y , z;  memset(pre , -1 , sizeof(pre));  memset(pp , -1 , sizeof(pp));  mcnt **=** bcnt **=** ncnt **=** cc **=** 0 **;**  while (m --)  {  scanf**(**"%d%d"**,&**x**,&**y**);**  e**[**mcnt**].**x **=** y **,** e**[**mcnt**].**next **=** pre**[**x**] ,** pre**[**x**] =** mcnt **++;**  e**[**mcnt**].**x **=** x **,** e**[**mcnt**].**next **=** pre**[**y**] ,** pre**[**y**] =** mcnt **++;**  }  memset**(**low **,** 0 **, sizeof(**low**));**  memset**(**DFN **,** 0 **, sizeof(**DFN**));**  memset**(**iscut **,** 0 **, sizeof(**iscut**));**  memset**(**bel **,** 0 **, sizeof(**bel**));**  memset(L , 0 , sizeof(L));  memset(id , 0 , sizeof(id));  memset(pel , 0 , sizeof(pel));  for (i = 1 ; i <= n ; ++ i)  if (!DFN[i])  tarjan**(**i **, -**1**);**  cutnum **=** bcnt**;**  for (i = 1 ; i <= n ; ++ i)  if (iscut[i])  id**[**i**] = ++** cutnum**;**  for (j = 1 ; j <= bcnt ; ++ j)  {  for (i = 0 ; i < bcc[j].size() ; ++ i)  if (iscut[y = bcc[j][i]])  {  y **=** id**[**y**] ,** x **=** j**;**  //cout << x << ' ' << y << endl;  g**[**cc**].**x **=** y **,** g**[**cc**].**next **=** pp**[**x**] ,** pp**[**x**] =** cc **++;**  g**[**cc**].**x **=** x **,** g**[**cc**].**next **=** pp**[**y**] ,** pp**[**y**] =** cc **++;**  }  }  memset(f , 0 , sizeof(f));  memset(vis , 0 , sizeof(vis));  for (i = 1 ; i <= cutnum ; ++ i)  if (!vis[i])  dfs**(**i **,** 0**);**  for (j = 1 ; 1 << j <= cutnum ; ++ j)  for (i = 1 ; i <= cutnum ; ++ i)  f**[**i**][**j**] =** f**[**f**[**i**][**j **-** 1**]][**j **-** 1**];**  scanf**(**"%d"**,&**m**);**  while (m --)  {  scanf**(**"%d%d"**,&**x **,&**y**);**  x **=** pel**[**x**] ,** y **=** pel**[**y**];**  //cout << x << ' ' << y << endl;  z **=** LCA**(**x **,** y**);**  printf**(**"%d\n" **, (**L**[**x**] +** L**[**y**] -** L**[**z**] -** L**[**z**]) >>** 1**);**  }  //cout << endl;  } |

### 2-SAT相关

|  |
| --- |
| 建图什么都很好说，重要在与输出一组方案：  **for** (i = 0 ; i < n << 1 ; ++ i)  **if** (!DFN[i]) **tarjan**(i);  **for** (i = 0 ; i < n ;i ++)  **if** (bel[i] == bel[i + n])  {  **puts**("NO");  **return**;  }//标记相反的节点所在连通分量  **else** cf[bel[i]] = bel[i + n] , cf[bel[i + n]] = bel[i];  **puts**("YES");  **for** (i = 0 ; i < n << 1 ; ++ i)  {  x = bel[i];//缩点建逆图  **for** (j = 0 ; j < e[i].**size**() ; ++ j)  {  y = bel[e[i][j]];  **if** (x != y)  g[y].**pb**(x) , ++ d[x] ;*// cerr << y << ' ' << x << endl;*  }  }  queue<int> q;//拓扑排序，从入度为0的点开始  **for** (i = 0 ;i < scnt ;i ++) **if** (!d[i]) q.**push**(i);  **while** (!q.**empty**())  {  x = q.**front**() , q.**pop**();  **if** (!f[x])//没有被标记则直接标记  f[x] = 1, f[cf[x]] = -1;  **for** (i = 0 ;i < g[x].**size**() ; ++ i)  **if** (!-- d[g[x][i]])  q.**push**(g[x][i]);  }  **for** (i = 0 ;i < n ; ++ i)  {  **if** (f[bel[i]] == 1)// 在这里f[bel[i]]的正负就是第i个变量在此方案的真假  x = a[i].fi , y = a[i].fi + b[i];  **else** x = a[i].se - b[i] , y = a[i].se;  **printf**("%02d:%02d %02d:%02d\n" , x/60 , x%60 , y/60 , y%60);  } |

### 生成树相关

|  |
| --- |
| MST具有环切性质。  MST利用边交换来求次小生成树等等  严格次小生成树   * 同最小生成树做法，但有略微不同。 * 先求出最小生成树MST * 然后枚举不在MST上的边(*u*,*v*)，若将(*u*,*v*)替换掉MST上节点*u*与节点*v*之间权值最大的边，若这条边的权值与*w*(*u*,*v*)相同，那么替换后的树不可能成为严格次小生成树。所以我们要替换节点*u*与节点v之间边权严格次大的边，这样得到的树才有可能是严格次小生成树。   MST和网络流有一定的联系  MST的Kruscal算法的过程要熟悉  最重要的是，MST是一棵树！ |

### 最小树形图（需更新

|  |
| --- |
| //mdst(n) ,n为root  int n , xx[N] , yy[N] , hh[N];  int d[N] , lim , ans , g[N][N] , eg[N] , q[N];  bool f[N] , pass[N] , more;  void combine(int id)  {    int tot = 0 , from , i , j , x;    for ( ; id  && !pass[id] ; id = eg[id])      q[tot ++] = id , pass[id] = 1;    for (from = 0 ; from < tot && q[from] != id ; ++ from);    if (from == tot) return;    more = 1;    for (i = from ; i < tot ; ++ i)    {      x = q[i];      ans += g[eg[x]][x];      if (i != from)      {        f[x] = 1;        for (j = 1 ; j <= n ; ++ j) if (!f[j])          g[id][j] = min(g[id][j] , g[x][j]);      }    }    for (i = 1 ; i <= n ; ++ i) if (!f[i] && i != id)      for (j = from ; j < tot ; ++ j)      {        x = q[j];        g[i][id] = min(g[i][id] , g[i][x] - g[eg[x]][x]);      }  }  int mdst(int root)  {    int i , j , k;    ans = 0;    memset(f , 0 ,sizeof(f));    for (more = 1 ; more ;)    {      more = 0;      memset(eg , 0 , sizeof(eg));      for (i = 1 ; i <= n ; ++ i) if (!f[i] && i != root)      {        for (j = 1 , k = 0 ; j <= n ; ++ j) if (!f[j] && i != j)          if (!k || g[j][i] < g[k][i])            k = j;        eg[i] = k;      }      memset(pass , 0 , sizeof(pass));      for (i = 1 ; i <= n ; ++ i) if (!f[i] && !pass[i] && i != root)        combine(i);    }    for (i = 1 ; i <= n ; ++ i)      if (!f[i] && i != root)        ans += g[eg[i]][i];    return ans;  }  void work()  {    int i , x , y , j;    ans = 0;    scanf("%d",&n);    for (i = 1 ; i <= n ; ++ i)      scanf("%d%d%d",&xx[i] , &yy[i] , &hh[i]);    scanf("%d%d%d%d", &lim ,&xx[n + 1] , &yy[n + 1] , &hh[n + 1]);    for (i = 1 ; i <= n ; ++ i)      g[n + 1][i] = abs(xx[n + 1] - xx[i]) + abs(yy[n + 1] - yy[i]) + abs(hh[n + 1] - hh[i]) , g[i][n + 1] = 1 << 30;    for (i = 1 ; i <= n ; ++ i)      for (j = 1 ; j <= n ; ++ j)        if (hh[j] <= lim)          g[i][j] = abs(xx[i] - xx[j]) + abs(yy[i] - yy[j]) + abs(hh[i] - hh[j]);        else g[i][j] = 1 << 30;    ++ n;    printf("%d\n" , mdst(n));  } |

### Bellman-Ford判负圈/差分约束系统（需更新

|  |
| --- |
| 判负圈，这份代码是计算平均值最小的回路的，需酌情修改。  queue换成stack效果会更好一些  bool **Bellman\_Ford**(double val)  {  deque<int> q;  int i , x , y ;double z;  **for** (i = 0 ; i < 676; ++ i)  d[i] = 0 , f[i] = 1 , q.**push\_back**(i), cnt[i] = 0;  **while** (!q.**empty**())  {  x = q.**front**() , q.**pop\_front**() , f[x] = 0;  **for** (i = e[x].**size**() - 1 ;i >= 0; -- i)  {  y = e[x][i].fi , z = (double)-e[x][i].se + val;  **if** (d[y] > d[x] + z)  {  d[y] = d[x] + z;  **if** (!f[y])  {  **if** (q.**empty**() || d[y] < d[q.**front**()])  q.**push\_front**(y);  **else** q.**push\_back**(y);  f[y] = 1; **if** (++ cnt[y] > 676) **return** 1;  }}}}  **return** 0;  }  三角不等式！  对所有边(u,v)满足d[u]+w(u,v)>=d[v]  假设有一个约束条件v – u <= w （u + w <= v）  就新建一条边u -> v，权值为 w  最后加一个超级源点和所有点连0边，跑Bellman-ford，有负圈则无解。 |

### Astar求K短路（需更新

|  |
| --- |
| int n , m , S , T , K;  struct edge {int x , w;};  vector<edge> e[N] , g[N];  int d[N];  bool operator < (edge x , edge y)  {  return x.w > y.w;  }  void \_Dijkstra(int st)  {  priority\_queue<edge> Q;  int i , x , y , z ;bool f[N] = {};  for (i = 1 ; i <= n ;++ i)  d[i] = 1 << 29;  d[st] = 0 ; Q.push((edge){st , 0});  while (!Q.empty())  {  edge a = Q.top(); Q.pop();  if (f[a.x]) continue;  f[x = a.x] = 1;  for (i = 0 ;i < g[x].size() ;++ i)  {  y = g[x][i].x , z = g[x][i].w;  if (!f[y] && d[x] + z < d[y])  d[y] = d[x] + z ,Q.push((edge){y , d[x] + z});  }  }  }  struct node { int x , w , c;};  bool operator < (node x , node y)  { return x.w > y.w;}  int Astar()  {  priority\_queue<node> q;  int i , x , cnt = 0;  q.push((node){S , d[S] , 0});  while (!q.empty())  {  node a = q.top() ; q.pop();  if ((x = a.x) == T)  cnt ++;  if (cnt == K)  return a.w;  for (i = 0 ;i < e[x].size() ;++ i)  q.push((node){e[x][i].x , a.c + d[e[x][i].x] + e[x][i].w, a.c + e[x][i].w});  }  return -1;  }  比较作死的直接吧edge当dist的类来用，实际情况一定要改一改……  当S和T相等的时候看看K是否要+1 |

### 全局最小割（需更新

|  |
| --- |
| 一个无向连通网络，去掉一个边集可以使其变成两个连通分量则这个边集就是割集；最小割集当然就权和最小的割集。 // 下标从0开始  int n , m , d[N] , g[N][N] , ans;  bool f[N] , u[N];  void **find**(int& s , int& t)  {  int i , j , x;  **memset**(u , 0 , **sizeof**(u));  **memset**(d , 0, **sizeof**(d));  **for** (i = 0 ;i < n ; ++ i)  {  x = -1;  **for** (j = 0 ;j < n ;j ++)  **if** (!u[j] && !f[j] && (d[j] > d[x] || x == -1))  x = j;  **if** (x == -1) {ans = **min**(ans , t == -1 ? 1 << 30 : d[t]); **return**;}  u[x] = 1 , s = t , t = x;  **for** (j = 0 ;j < n ;j ++)  **if** (!u[j] && !f[j])  d[j] += g[x][j];  }  ans = **min**(ans , d[t]);  }  void **work**()  {  int i , j , x , y , z; ans = 1 << 30;  **memset**(g , 0, **sizeof**(g));  **for** (i = 1 ; i <= m ;i ++)  {  **scanf**("%d%d%d",&x,&y,&z);  g[x][y] += z , g[y][x] += z;  }  **memset**(f , 0, **sizeof**(f));  **for** (i = 1 ; i < n ;i ++)  {  x = y = -1 , **find**(x , y);  f[y] = 1;  **for** (j = 0 ; j < n ; ++ j)  g[x][j] += g[y][j] , g[j][x] += g[j][y];  }  cout << ans << endl;  } |

### 倍增LCA

|  |
| --- |
| int LCA(int x , int y)  {  if (L[x] > L[y]) swap(x , y);  for (i = 16 ; i >= 0 ; -- i)  if (L[y] - L[x] >= 1 << i)  y = f[i][y];  if (x == y) return y;  for (i = 16 ; i >= 0 ; -- i)  if (f[i][x] && f[i][x] != f[i][y])  x = f[i][x] , y = f[i][y];  return f[0][x];  } |

### Hopcroft-Karp

|  |
| --- |
| 最大独立集/最小覆盖集/最大匹配的关系 int mx[N] , my[N];  queue<int> que;  int dx[N] , dy[N];  bool vis[N];  bool find(int x)  {      for (int i = pre[x] ; ~i ; i = e[i].next)      {          int y = e[i].x;          if (!vis[y] && dy[y] == dx[x] + 1)          {              vis[y] = 1;              if (!~my[y] || find(my[y]))              {                  mx[x] = y , my[y] = x;                  return 1;              }          }      }      return 0;  }  int matching()  {      memset(mx , -1 , sizeof(mx));      memset(my , -1 , sizeof(my));      int ans = 0;      while (1){          bool flag = 0;          while (!que.empty()) que.pop();          memset(dx , 0 , sizeof(dx));          memset(dy , 0 , sizeof(dy));          for (int i = 0 ; i < n ; ++ i)              if (!~mx[i]) que.push(i);          while (!que.empty())          {              int x = que.front(); que.pop();              for (int i = pre[x] ; ~i ; i = e[i].next)              {                  int y = e[i].x;                  if (!dy[y])                  {                      dy[y] = dx[x] + 1 ;                      if (~my[y])                          que.push(my[y]) , dx[my[y]] = dy[y] + 1;                      else                          flag = 1;                  }              }          }          if (!flag) break;          memset(vis , 0 , sizeof(vis));          for (int i = 0 ; i < n ; ++ i)              if (!~mx[i] && find(i)) ++ ans;      }      return ans;  } |

### Dinic

|  |
| --- |
| #define N 20005  #define M 2000005  using namespace std;  int n , m , s , t;  struct arc  {  int x , f , next;  }e[M];  int pre[N] , mcnt;  void addarc(int x ,int y ,int z)  {  e[mcnt] = (arc) {y , z , pre[x]} , pre[x] = mcnt ++;  e[mcnt] = (arc) {x , z , pre[y]} , pre[y] = mcnt ++;  }  int d[N] , cur[N] , q[N];  bool BFS()  {  memset(d , 0 , sizeof(int) \* (t + 1));  int top = 0 , bot = -1;  q[++ bot] = s , d[s] = 1;  while (top != bot + 1)  {  int x = q[top ++];  for (int i = pre[x] ; ~i ;i = e[i].next)  {  int y = e[i].x;  if (!d[y] && e[i].f)  {  d[y] = d[x] + 1 , q[++ bot] = y;  if (y == t) return 1;  }  }  }  return 0;  }  int DFS(int x , int flow = 1 << 30)  {  if (x == t || !flow) return flow;  int sum = 0 , u;  for (int& i = cur[x] ; ~i ; i = e[i].next)  {  int y = e[i].x;  if (d[y] == d[x] + 1 && (u = DFS(y , min(flow , e[i].f))))  {  e[i].f -= u , e[i ^ 1].f += u;  sum += u , flow -= u;  if (!flow) break;  }  }  if (sum == flow) d[x] = -1;  return sum;  }  int dinic()  {  int ans = 0;  while (BFS())  {  memcpy(cur , pre , (t + 1) \* sizeof(int));  ans += DFS(s);  }  return ans;  } |

### ISAP

|  |
| --- |
| #define N 20005  #define M 2000005  int n , m , nodecnt;  int s , t , pre[N] , mcnt;  struct arc  {  int x , f , next;  }e[M];  void addarc(int x ,int y ,int z)  {  e[mcnt] = (arc) {y , z , pre[x]} , pre[x] = mcnt ++;  e[mcnt] = (arc) {x , z , pre[y]} , pre[y] = mcnt ++;  }  int q[N] , d[N] , cur[N] , gap[N] , neck[N] , backup[N] , p[N];  void BFS()  {  int i , x , top = 0 , bot = -1;  memset(gap , 0 , sizeof(gap));  for (i = 1 ;i <= nodecnt ; i ++)  d[i] = nodecnt + 1;  q[++ bot] = t , d[t] = 0;  while (top != bot + 1)  {  x = q[top ++];  for (i = pre[x] ; ~i ; i = e[i].next)  {  int y = e[i].x;  if (e[i ^ 1].f && d[y] == nodecnt + 1)  {  d[y] = d[x] + 1 , ++ gap[d[y]];  q[++ bot] = y;  }  }  }  }  int SAP()  {  BFS();  int x = s , i , j , y , maxflow = 0 , flow = 1 << 30;  memcpy(cur , pre , sizeof(int) \* (nodecnt + 1));  neck[s] = s;  while (d[s] < nodecnt)  {  backup[x] = flow;  for (i = cur[x] ; ~i ; i = e[i].next)  if (d[x] == d[e[i].x] + 1 && e[i].f)  break;  if (~i)  {  cur[x] = i;  y = e[i].x;  if (e[i].f < flow)  flow = e[i].f , neck[y] = x;  else  neck[y] = neck[x];  p[y] = i , x = y;  if (x == t)  {  maxflow += flow;  for (i = x ; i != s ; i = e[p[i] ^ 1].x)  e[p[i]].f -= flow , backup[i] -= flow , e[p[i] ^ 1].f += flow;  x = neck[x] , flow = backup[x];  }  }  else  {  j = -1;  for (i = pre[x] ; ~i ;i = e[i].next)  if (e[i].f && (!~j || d[e[i].x] < d[e[j].x]))  j = i;  if (!~j || !-- gap[d[x]]) break;  cur[x] = j;  ++ gap[d[x] = d[e[j].x] + 1];  if (x != s)  x = e[p[x] ^ 1].x , flow = backup[x];  }  }  return maxflow;  } |

### 费用流（bellman\_ford式）

|  |
| --- |
| #define N 5005  int n , m , s , t;  int pre[N] , mcnt;  struct arc {  int x , f , c , next;  }e[200005];  void addarc(int x ,int y ,int z ,int c)  {  //printf("%d %d %d %d\n" , x , y , z , c);  e[mcnt] = (arc){y , z , c , pre[x]} , pre[x] = mcnt ++;  e[mcnt] = (arc){x , 0 , -c , pre[y]} , pre[y] = mcnt ++;  }  int d[N] , p[N];  bool f[N];  deque<int> q;  bool Bellman\_Ford()  {  int i , x , y , z;  for (i = 0 ; i <= t ; ++ i)  d[i] = 1 << 30 , f[i] = 0;  d[s] = 0 , f[s] = 1 , q.push\_back(s);  while (!q.empty()) {  x = q.front() , q.pop\_front() , f[x] = 0;  for (i = pre[x] ; ~i ; i = e[i].next) {  y = e[i].x , z = e[i].c;  if (e[i].f && d[y] > d[x] + z) {  d[y] = d[x] + z , p[y] = i;  if (!f[y]) {  if (q.empty() || d[y] < d[q.front()])  q.push\_front(y);  else q.push\_back(y);  f[y] = 1;  }  }  }  }  return d[t] != 1 << 30;  }  int Mincostflow()  {  int maxflow = 0 , ans = 0;  int x;  while (Bellman\_Ford()) {  int flow = 1 << 30;  for (x = t ; x != s ; x = e[p[x] ^ 1].x)  flow = min(flow , e[p[x]].f);  maxflow += flow , ans += d[t] \* flow;  for (x = t ; x != s ; x = e[p[x] ^ 1].x)  e[p[x]].f -= flow , e[p[x] ^ 1].f += flow;  }  return ans;  } |

### 费用流（zkw式）（需更新

|  |
| --- |
| int s , t , ans , maxflow;  int pre[N] , mcnt;  struct arc  {    int x , f , c , next;  }e[N \* N];  void addarc(int x ,int y ,int z ,int c)  {    e[mcnt] = (arc){y , z , c , pre[x]} , pre[x] = mcnt ++;    e[mcnt] = (arc){x , 0 , -c , pre[y]} , pre[y] = mcnt ++;    // printf("%d %d %d %d\n",x,y,z,c);  }  int d[N];  bool f[N];  queue<int> q;  void Bellman\_Ford()  {    int i , x , y , z;    memset(f , 0 , sizeof(f));    for (i = 0; i <= t ; ++ i) d[i] = 1 << 30;    d[s] = 0 , f[s] = 1 , q.push(s);    while (!q.empty()){      x = q.front() , q.pop() , f[x] = 0;      for (i = pre[x] ; ~i ; i = e[i].next){        y = e[i].x , z = e[i].c;        if (e[i].f && d[y] > d[x] + z){          d[y] = d[x] + z;          if (!f[y])            q.push(y) , f[y] = 1;          }}}    for (i = 0; i <= t ; ++ i) d[i] = d[t] - d[i];  }  int zkwpush(int x , int flow = 1 << 30)  {    if (x == t || !flow)    {      maxflow += flow , ans += d[s] \* flow;      return flow;    }    f[x] = 1; int now = flow;    for (int i = pre[x] ; ~i ; i = e[i].next)    {      int y = e[i].x , u;      if (e[i].f && !f[y] && d[y] + e[i].c == d[x] && (u = zkwpush(y , min(now , e[i].f))))      {        e[i].f -= u , e[i ^ 1].f += u;        now -= u; if (!now) break;      }    }    return flow - now;  }  bool modify()  {    int dis = 1 << 30;    for (int x = 0 ; x <= t ; ++ x) if (f[x])      for (int i = pre[x] ; ~i ; i = e[i].next)      {        int y = e[i].x;        if (e[i].f && !f[y]) dis = min(dis , d[y] + e[i].c - d[x]);      }    if (dis == 1 << 30) return 0;    for (int x = 0 ; x <= t ; ++ x) if (f[x]) d[x] += dis;    return 1;  }  pair<int , int> Mincostflow()  {    maxflow = 0 , ans = 0;    Bellman\_Ford();    while (1) {while (1){      memset(f , 0 , sizeof(f));      if (!zkwpush(s)) break;    }if (!modify()) break;}    return make\_pair(ans , maxflow);  } |

### 网络流建模Tips

|  |
| --- |
| ①有上下界网络流  设原来的源点为S，汇点为T，新建SS，ST  对u到v的每条边，容量范围[L,U], 拆成3条  u->ST , L , SS->v , L , u->v U-L  最后添加T->S，inf 求SS到ST的最大流，所有从SS出发的弧都满流即存在可行流  注：必要时合并SS出发和到达ST的弧  最小流：添加T->S前增广一次，添加后增广的流量即最小流  最大流：还原到原来网络，直接增广。  ②二分图带权最大独立集（最小割）  最喜闻乐见的建图模型，带权最小覆盖集也一样  ③最大权闭合子图  从s向所有正权点连正边，所有负权点向t连正边，求最小割。  正权总和-最小割为答案  模型隐蔽，注意依赖关系、拓扑排序去环等。  ④区间k覆盖问题  数轴上一堆带权左闭右开区间，求权和最大的一堆区间使得任意一个数最多被k个区间覆盖。  把所有数看做点，一个区间([u,v),w)加边u->v , 1 , w  相邻点加i->i+1，k,0。从左到右求最大费用流，必要时离散化。  ⑤费用流和圈有一点关系 |

### 一般图匹配

|  |
| --- |
| int f[N] , match[N] , bel[N] , mark[N];  bool u[N] , InBlossom[N] , g[N][N];  int Q[N] , top , bot;  void BlossomContract(int x , int y)  {  memset(mark , 0 , sizeof(mark));  memset(InBlossom , 0 , sizeof(InBlossom));  #define pre f[match[i]]  int lca,i;  for (i = x ; i ; i = pre) i = bel[i] , mark[i] = 1;  for (i = y ; i ; i = pre)  {  i = bel[i];  if (mark[i])  {  lca = i;  break;  }  }  for (i = x ; bel[i] != lca ; i = pre)  {  if (bel[pre] != lca) f[pre] = match[i];  InBlossom[bel[i]] = 1;  InBlossom[bel[match[i]]] = 1;  }  for (i = y ; bel[i] != lca ; i = pre)  {  if (bel[pre] != lca) f[pre] = match[i];  InBlossom[bel[i]] = 1;  InBlossom[bel[match[i]]] = 1;  }  if (bel[x] != lca) f[x] = y; //注意不能从lca这个奇环的关键点跳回来  if (bel[y] != lca) f[y] = x;  for (i = 1 ; i <= n ; ++ i)  if (InBlossom[bel[i]])  {  bel[i] = lca;  if (!u[i]){  Q[++ bot] = i;  u[i] = 1; //要注意如果本来连向BFS树 }  } 中父结点的边是非匹配边的点，可能是没有入队的  }  void Change(int z)  {  int x , y ;  while (z)  {  y = f[z] , x = match[y];  match[y] = z , match[z] = y;  z = x;  }  }  void find(int st)  {  int i , x , y;  memset(f , 0 , sizeof(f));  memset(u , 0 , sizeof(u));  for (i = 1 ; i <= n ; ++ i)  bel[i] = i;  top = 0 , bot = -1;  Q[++ bot] = st , u[st] = 1;  while (top != bot + 1)  {  x = Q[top ++];  for (y = 1 ; y <= n ; ++ y)  if (g[x][y] && bel[x] != bel[y] && match[x] != y)  if (st == y || match[y] && f[match[y]])  BlossomContract(x , y);  else if (!f[y])  {  f[y] = x;  if (match[y])  Q[++ bot] = match[y] , u[match[y]] = 1;  else  {  Change(y);  return;  }  }  }  } |

### Tuttx矩阵求一般图匹配数

|  |
| --- |
| int g[N][N] , a[N][N];  pair<int , int> e[105];  **int** inverse**(int** x**)**  {  return x == 1 ? 1 : (long long)(MOD - MOD / x) \* inverse(MOD % x) % MOD;  }  int Match()  {  for (int i = 0; i < n; ++ i) {  for (int j = 0; j < n; ++ j) {  if (a[i][j]) {  a**[**i**][**j**] =** rand**() %** MOD**;**  a**[**j**][**i**] =** MOD **-** a**[**i**][**j**];**  }  }  }  int rank = 0;  for (int i = 0; i < n; ++ i) {  int pivot = rank;  while (pivot < n && !a[pivot][i]) {  pivot **++;**  }  if (pivot < n) {  for (int j = 0; j < n; ++ j) {  swap**(**a**[**rank**][**j**],** a**[**pivot**][**j**]);**  }  {  int times = inverse(a[rank][i]);  for (int j = 0; j < n; ++ j) {  a**[**rank**][**j**] = (long long)**a**[**rank**][**j**] \*** times **%** MOD**;**  }  for (int k = 0; k < n; ++ k) {  if (k != rank && a[k][i]) {  int times = a[k][i];  for (int j = 0; j < n; ++ j) {  (a[k][j] += MOD - (long long)a[rank][j] \* times % MOD) %= MOD;  }  }  }  }  rank **++;**  }  }  return rank >> 1;  } |

### 二分图最优匹配KM（需更新

|  |
| --- |
| int n , g[N][N] , lx[N] , ly[N] , match[N] , ans , slack[N];  bool fx[N] , fy[N];  bool find(int x)  {    fx[x] = 1;    for (int y = 1 ;y <= n ;y ++)      if (lx[x] + ly[y] == g[x][y] && !fy[y])      {        fy[y] = 1;        if (!match[y] || find(match[y]))        {          match[y] = x;          return 1;        }      }      else if (lx[x] + ly[y] != g[x][y] )        slack[y] = min(slack[y] , lx[x] + ly[y] - g[x][y]);    return 0;  }  void update()  {    int a = 1 << 30 , i;    for (i = 1 ; i <= n ;i ++)      if (!fy[i])        a = min(a, slack[i]);    for (i = 1 ; i <= n ;i ++)    {      if (fx[i]) lx[i] -= a;      if (fy[i]) ly[i] += a;      slack[i] -= a;    }  }  void Kuhn\_Munkras()  {    int i , j;    for (i = 1 ;i <= n ;i ++)    {      match[i] = lx[i] = ly[i] = 0;      for (j = 1 ; j <= n ;j ++)        lx[i] = max(lx[i] , g[i][j]);    }    for (i = 1 ;i <= n ;i ++)      while (1)      {        for (j = 1 ;j <= n ;j ++)          fx[j] = fy[j] = 0, slack[j] = 1 << 30;        if (find(i))          break;        else update();      }    for (i = 1 ;i <= n ;i ++)      ans += lx[i] + ly[i];  } |

### 最大团最大独立集等知识

|  |
| --- |
| 找一个最大团  int n , mc[N] , list[N][N] , len[N] , ans;  bool g[N][N] , found;  void dfs(int size)  {  int i , j , k;  if (!len[size])  {  if (size > ans)  ans = size , found = 1;  return;  }  for (k = 0 ; k < len[size] && !found ; ++ k)  {  if (size + len[size] - k <= ans)  break;  i = list[size][k];  if (size + mc[i] <= ans)  break;  for (j = k + 1 , len[size + 1] = 0 ; j < len[size] ; ++ j)  if (g[i][list[size][j]])  list[size + 1][len[size + 1] ++] = list[size][j];  dfs(size + 1);  }  }  void max\_cluster()  {  int i , j;  mc[n] = ans = 1;  for (i = n - 1 ; i ; -- i)  {  found = 0 , len[1] = 0;  for (j = i + 1 ; j <= n ; ++ j)  if (g[i][j])  list[1][len[1] ++] = j;  dfs(1);  mc[i] = ans;  }  }  void work()  {  for (int i = 1 ; i <= n ; ++ i)  for (int j = 1 ; j <= n ; ++ j)  scanf("%d",&g[i][j]);  max\_cluster();  cout << ans << endl;  }  最大独立集即补图的最大团 |

## 数学

### 分数类

|  |
| --- |
| struct frac  {  LL a , b;  frac() { a = 0 , b = 1;}  frac(LL \_a , LL \_b) {a = \_a , b = \_b;}  void maintain()  {  long long G = \_\_gcd(a , b);  a /= G , b /= G;  if (b < 0) a = -a , b = -b;  }  bool operator < (const frac& R) const{  return a \* R.b < b \* R.a;  }  frac operator + (const frac& R) const{  frac res(a \* R.b + b \* R.a , R.b \* b);  res.maintain();  return res;  }  frac operator - (const frac& R) const{  frac res(a \* R.b - b \* R.a , R.b \* b);  res.maintain();  return res;  }  frac operator \* (const frac& R) const{  frac res(a \* R.a , R.b \* b);  res.maintain();  return res;  }  frac operator / (const frac& R) const{  frac res(a \* R.b , R.a \* b);  res.maintain();  return res;  }  frac& operator - ()  {  a = -a;  return \*this;  }  frac& operator ! ()  {  swap(a , b);  if (b < 0) a = -a , b = -b;  return \*this;  }  LL operator = (LL R)  {  a = R , b = 1;  return R;  }  }; |

### 矩阵类

|  |
| --- |
| struct Matrix  {  int n , m ;  double a[N][N];  Matrix() {memset(a , 0 , sizeof(a));}  Matrix(int \_n , int \_m) {  n = \_n , m = \_m , memset(a , 0 , sizeof(a));  };  double\* operator[] (int i) {  return a[i];  }  };  Matrix operator \* (Matrix A , Matrix B)  {  Matrix ans(A.n , B.m);  for (int i = 0 ; i < A.n ; ++ i)  for (int j = 0 ; j < A.m ; ++ j)  for (int k = 0 ; k < B.m ; ++ k)  ans[i][k] += A[i][j] \* B[j][k];  return ans;  } |

### FFT

|  |
| --- |
| #define N 65536  #define M 4000005  struct Complex  {  double x , y;  Complex () {x = y = 0;}  Complex (double \_x , double \_y) {x = \_x , y = \_y;}  Complex operator + (const Complex& r) const {  return Complex(x + r.x , y + r.y);  }  Complex operator - (const Complex& r) const {  return Complex(x - r.x , y - r.y);  }  Complex operator \* (const Complex& r) const {  return Complex(x \* r.x - y \* r.y , x \* r.y + y \* r.x);  }  int operator = (const int a) {  \*this = Complex(a , 0);  return a;  }  };  const double pi = acos(-1.0);  void FFT(Complex P[], int n, int oper)  {  for (int i = 1, j = 0; i < n - 1; i++) {  for (int s = n; j ^= s >>= 1, ~j & s;);  if (i < j) {  swap(P[i], P[j]);  }  }  for (int d = 0; (1 << d) < n; d++) {  int m = 1 << d, m2 = m \* 2;  double p0 = pi / m \* oper;  Complex unit\_p0(cos(p0) , sin(p0));  for (int i = 0; i < n; i += m2) {  Complex unit(1 , 0);  for (int j = 0; j < m; j++) {  Complex &P1 = P[i + j + m], &P2 = P[i + j];  Complex t = unit \* P1;  P1 = P2 - t;  P2 = P2 + t;  unit = unit \* unit\_p0;  }  }  }  }  Complex A[N] , B[N];  char a[N] , b[N];  int len , C[N];  void work()  {  int i , len1 = strlen(a) , len2 = strlen(b);  int n = 0 , m = 0 , x , y = 0;  for (i = len1 - 1 ; i >= 0 ; i -= 2)  {  x = 0;  if (i >= 0) x += (a[i] - '0');  if (i >= 1) x += (a[i - 1] - '0') \* 10;  A[n ++] = x;  }  for (i = len2 - 1 ; i >= 0 ; i -= 2)  {  x = 0;  if (i >= 0) x += (b[i] - '0');  if (i >= 1) x += (b[i - 1] - '0') \* 10;  B[m ++] = x;  }  len = 1;  while (len < n + m - 1) len <<= 1;  for (i = n ; i < len ; ++ i) A[i] = 0;  for (i = m ; i < len ; ++ i) B[i] = 0;  FFT(A , len , 1);  FFT(B , len , 1);  for (i = 0 ; i < len ; ++ i)  A[i] = A[i] \* B[i];  FFT(A , len , -1);  for (i = 0 ; i < len ; ++ i)  {  C[i] = A[i].x / len + 0.5 + y;  y = C[i] / 100 , C[i] %= 100;  }  for (i = len - 1 ; i >= 1 && !C[i] ; -- i);  printf("%d" , C[i]);  for ( -- i ; i >= 0 ; -- i)  printf("%02d" , C[i]);  puts("");  }  int main()  {  while (~scanf("%s%s", a, b))  work();  return 0;  } |

### 自适应Simpson数值积分

|  |
| --- |
| //Adaptive Simpson's Rule integral  double F(double x)  {  return sqrt(1 + 4 \* a \* a \* x \* x);  }  double simpson(double a , double b)  {  double c = (a + b) \* 0.5;  return (F(a) + 4 \* F(c) + F(b)) \* (b - a) / 6;  }  double asr(double a , double b , double eps , double A)  {  double c = (a + b) \* 0.5;  double L = simpson(a , c) , R = simpson(c , b);  if (fabs(L + R - A) <= 15 \* eps)  return L + R + (L + R - A) / 15;  return asr(a , c , eps / 2 , L) + asr(c , b , eps / 2 , R);  }  double cal()  {  a = 4 \* m / w / w;  return n \* 2 \* asr(0 , w / 2 , 1e-5 , simpson(0 , w / 2));  } |

## 几何

### 2D（未完成

|  |
| --- |
| const double eps = 1e-8 , pi = cos(-1.0);  int dcmp(double x) {  return fabs(x) < eps ? 0 : x < 0 ? -1 : 1;  }  struct Point  {  double x , y;  Point () {x = y = 0;}  Point (double \_x , double \_y) {x = \_x , y = \_y;}  void input() {  scanf("%lf%lf",&x,&y);  }  void output() {  printf("%f %f\n", x , y);  }  bool operator < (const Point& R) const{  return x < R.x || (x == R.x && y < R.y);  }  bool operator == (const Point& R) const{  return dcmp(x - R.x) == 0 && dcmp(y - R.y) == 0;  }  Point operator + (const Point& R) const{  return Point(x + R.x , y + R.y);  }  Point operator - (const Point& R) const{  return Point(x - R.x , y - R.y);  }  Point operator \* (const double& R) const{  return Point(x \* R , y \* R);  }  Point operator / (const double& R) const{  return Point(x / R , y / R);  }  double operator ^ (const Point& R) const{  return x \* R.y - y \* R.x;  }  double operator % (const Point& R) const{  return x \* R.x + y \* R.y;  }  double len()  {  return sqrt(\*this % \*this);  }  };  double Angle(Point A , Point B)  {  return acos((A % B) / A.len() / B.len());  }  Point Rotate(Point A , double rad)  {  double Sin = sin(rad) , Cos = cos(rad);  return Point(A.x \* Cos - A.y \* Sin , A.x \* Sin + A.y \* Cos);  }  Point Normal(Point A)  {  double L = A.len();  return Point(-A.y / L , A.x / L);  }  Point GetLineIntersection(Point P , Point v , Point Q , Point w)  {  Point u = P - Q;  double t1 = (w ^ u) / (v ^ w);  return P + v \* t1;  }  double DistancePointToLine(Point P , Point A , Point B)  {  Point v = B - A;  return (v ^ (P - A)) / v.len();  }  double DistancePointToSegment(Point P , Point A , Point B)  {  if (A == B) return (P - A).len();  Point v1 = B - A , v2 = P - A , v3 = P - B;  if (dcmp(v1 % v2) < 0) return v2.len();  if (dcmp(v1 % v3) > 0) return v3.len();  return fabs(v1 ^ v2) / v1.len();  }  Point GetLineProjection(Point P , Point A , Point B)  {  Point v = B - A;  return A + v \* (v % (P - A) / (v % v));  }  bool SegmentProperIntersection(Point a1 , Point a2 , Point b1 , Point b2)  {  double c1 = (a2 - a1) ^ (b1 - a1);  double c2 = (a2 - a1) ^ (b2 - a1);  double c3 = (b2 - b1) ^ (a1 - b1);  double c4 = (b2 - b1) ^ (a2 - b1);  return dcmp(c1) \* dcmp(c2) < 0 && dcmp(c3) \* dcmp(c4) < 0;  }  bool OnSegment(Point P , Point a1 , Point a2)  {  return dcmp((a1 - P) ^ (a2 - P)) == 0 && dcmp((a1 - P) % (a2 - P)) < 0;  }  struct Line  {  Point P , V; // P + Vt  Line () {}  Line (Point A , Point B) {P = A , V = B - A;}  Point point(double t){  return Point(P.x + V.x \* t , P.y + V.x \* t);  }  };  struct Circle  {  Point C;  double r;  Circle () {}  Circle (Point \_C , double \_r) {C = \_C , r = \_r;}  Point point(double arc){  return Point(C.x + cos(arc) \* r , C.y + sin(arc) \* r);  }  };  int getLineCircleIntersection(Line L , Circle C , double& t1 , double& t2 , vector<Point>& sol)  {  double a = L.V.x , b = L.P.x - C.C.x , c = L.V.y , d = L.P.y - C.C.y;  double e = a \* a + c \* c , f = 2 \* (a \* b + c \* d) , g = b \* b + d \* d - C.r \* C.r;  double delta = f \* f - 4 \* e \* g;  if (dcmp(delta) < 0) return 0;  if (dcmp(delta) == 0)  {  t1 = t2 = -f / (2 \* e);  sol.push\_back(L.point(t1));  return 1;  }  t1 = (-f - sqrt(delta)) / (e + e);  t2 = (-f + sqrt(delta)) / (e + e);  sol.push\_back(L.point(t1)) , sol.push\_back(L.point(t2));  return 2;  }  double angle(Point V)  {  return atan2(V.y , V.x);  }  int getCircleCircleIntersection(Circle C1 , Circle C2 , vector<Point>& sol)  {  double d = (C1.C - C2.C).len();  if (dcmp(d) == 0)  {  if (dcmp(C1.r - C2.r) == 0)  return -1;  return 0;  }  if (dcmp(C1.r - C2.r - d) < 0) return 0;  if (dcmp(fabs(C1.r - C2.r) - d) > 0) return 0;  double a = angle(C2.C - C1.C);  double da = acos((C1.r \* C1.r + d \* d - C2.r \* C2.r) / (2 \* C1.r \* d));  Point P1 = C1.point(a - da) , P2 = C1.point(a + da);  sol.push\_back(P1);  if (P1 == P2) return 1;  sol.push\_back(P2);  return 2;  }  // 过点p到圆C的切线。返回切线条数  int getTangents(Point P, Circle C, vector<Point>& sol)  {  Point u = C.C - P;  double dist = u.len();  if(dist < C.r) return 0;  if(dcmp(dist - C.r) == 0)  { // p在圆上，只有一条切线  sol.push\_back(Rotate(u, pi/2));  return 1;  } else {  double ang = asin(C.r / dist);  sol.push\_back(Rotate(u, +ang));  sol.push\_back(Rotate(u, -ang));  return 2;  }  } |

## 字符串

### KMP

|  |
| --- |
| void getFail(char \*P)  {    int m = strlen(P);    f[0] = f[1] = 0;    for (int i = 1 ; i < m ;i ++)    {      int j = f[i];      while (j && P[i] != P[j]) j = f[j];      f[i + 1] = P[i] == P[j] ? j + 1 : 0;    }  }  void find(char\* T , char\* P , int\* u)  {    m = strlen(P);    for (int i = 0 ,j = 0 ; T[i] ;i ++) {      while (j && T[i] != P[j]) j = u[j];      if (T[i] == P[j]) ++ j;      if (j == m) printf("%d\n",i - m + 1);    }  } |

### 扩展kmp

|  |
| --- |
| 对P自我匹配求出每个后缀与本身的LCP  就可以递推再求出T的每个后缀与P的LCP  j = 0;  while (j + 1 < l[x] && s[x][j] == s[x][j + 1]) ++ j;  fail[1] = j , k = 1;  for (i = 2 ; i < l[x] ; ++ i)  {  int len = k + fail[k] - 1, L = fail[i - k];  if (L <= len - i)  fail[i] = L;  else  {  j = max(0 , len - i + 1);  while(i + j < l[x] && s[x][j] == s[x][i + j]) ++ j;  fail[i] = j , k = i;  }  }  j = 0;  while (j < l[x] && j < l[y] && s[x][j] == s[y][j]) ++ j;  ret[0] = j , k = 0;  for (i = 1 ; i < l[y] ; ++ i)  {  int len = k + fail[k] - 1 , L = fail[i - k];  if (L <= len - i)  ret[i] = L;  else  {  j = max(0 , len - i + 1);  while (i + j < l[y] && j < l[x] && s[x][j] == s[y][j + i]) ++ j;  ret[i] = j , k = i;  }  } |

### Manacher

|  |
| --- |
| scanf("%d%d%s" , &n , &m , str);  s[0] = '%';  **for** (i = 0 ; str[i] ; ++ i)      s[len ++] = '#' , s[len ++] = str[i]; s[len ++] = '#';s[len] = 0;  **int** id = 0 , mx = 0;  **for** (i = 0 ; i != len ; ++ i)  {      p[i] = mx > i ? min(p[id + id - i], mx - i) : 1;  **while** (s[i + p[i]] == s[i - p[i]]) ++ p[i];  **if** (i + p[i] > mx)        mx = i + p[i] , iid = i;  } |

### 最小表示法

|  |
| --- |
| int MinR(char \*str)//MinimumRepresentation  {  int i = 0 , j = 1 , k = 0 , len = strlen(str);  while (i < len && j < len && k < len)  {  int cmp = str[(j + k) % len] - str[(i + k) % len];  if (!cmp)  ++ k;  else  {  if (cmp > 0)  j += k + 1;  else i += k + 1;  if (i == j) ++ j;  k = 0;  }  }  return min(i , j);  } |

### AC自动机

|  |
| --- |
| 将AC自动机中的Fail指针反向之后得到一棵树  则自动机中某个字串x在另一个字串y中的出现个数可以转化为：  从位置0到字串y的位置所对应的所有端点，有多少在字串x对应端点在Fail树中的子树里。  只要用树状数组维护Fail树的dfs序列，就能回答这些询问了。  回答的时候离线，按AC自动机上的dfs序挨个访问节点。  struct ACautomaton  {  int nodecnt;  int u[N][26] , f[N];  five val[N];  queue<int> q;  int newnode() {  val[nodecnt] = five();  memset(u[nodecnt] , 0 , sizeof(u[nodecnt]));  return nodecnt ++;  }  void clear() {  nodecnt = 0;  newnode();  }  void insert(char str[] , int id) {  int x = 0;  for (int i = 0 ; str[i] ; ++ i) {  int c = str[i] - 'a';  if (!u[x][c])  u[x][c] = newnode();  x = u[x][c];  }  val[x].insert(id);  }  int query(char \*str) {  int x = 0;  for (int i = 0 ; str[i] ; ++ i) {  int c = str[i] - 'a';  x = u[x][c] , val[x].query();  }  }  void getFail() {  for (int i = 0 ; i < 26 ; ++ i)  if (u[0][i])  f[u[0][i]] = 0 , q.push(u[0][i]);  while (!q.empty()) {  int x = q.front() ; q.pop();  for (int i = 0 ; i < 26 ; ++ i) {  int y = u[x][i];  if (!y) {u[x][i] = u[f[x]][i]; continue;}  q.push(y); int j = f[x];  while (j && !u[j][i]) j = f[j];  f[y] = u[j][i] , val[y].update(val[f[y]]);  }  }  }  }heap; |

### 后缀数组

|  |
| --- |
| #define N 50005  int n , m , K; char s[N];  int sa[N] , t1[N] , t2[N] , c[N];  int rank[N] , height[N];  int f[16][N] , LOG[N];  void buildsa(int m)  {  int i , k , \*x = t1 , \*y = t2;  for (i = 0 ; i < m ; ++ i) c[i] = 0;  for (i = 0 ; i < n ; ++ i) ++ c[x[i] = s[i]];  for (i = 1 ; i < m ; ++ i) c[i] += c[i - 1];  for (i = n - 1 ; i >= 0 ; -- i) sa[-- c[x[i]]] = i;  for (k = 1 ; k <= n ; k <<= 1)  {  int p = 0;  for (i = n - k ; i < n ; ++ i) y[p ++] = i;  for (i = 0 ; i < n ; ++ i) if (sa[i] >= k) y[p ++] = sa[i] - k;  for (i = 0 ; i < m ; ++ i) c[i] = 0;  for (i = 0 ; i < n ; ++ i) ++ c[x[y[i]]];  for (i = 1 ; i < m ; ++ i) c[i] += c[i - 1];  for (i = n - 1 ; i >= 0 ; -- i) sa[-- c[x[y[i]]]] = y[i];  swap(x , y) , p = 1 , x[sa[0]] = 0;  for (i = 1 ; i < n ; ++ i)  x[sa[i]] = (y[sa[i-1]] == y[sa[i]] && y[sa[i-1]+k] == y[sa[i]+k]) ? p - 1 : p ++;  if (p >= n) break;  m = p;  }  k = 0;  for (i = 0 ; i < n ; ++ i) rank[sa[i]] = i;  for (i = 0 ; i < n ; ++ i)  {  if (k) -- k; if (!rank[i]) continue;  int j = sa[rank[i] - 1];  while (s[i + k] == s[j + k]) ++ k;  height[rank[i]] = k;  }  memset(f , 0 , sizeof(f));  for (i = 0 ; i < n ; ++ i)  f[0][i] = height[i];  for (int j = 1 ; 1 << j <= n ; ++ j)  for (i = 1 ; i + (1 << j) - 1 < n ; ++ i)  f[j][i] = min(f[j - 1][i] , f[j - 1][i + (1 << j - 1)]);  }  int LCP(int x , int y)  {  x = rank[x] , y = rank[y];  if (x > y) swap(x , y);  ++ x;  int j = LOG[y - x + 1];  return min(f[j][x] , f[j][y - (1 << j) + 1]);  }  //buildsa时需要让字符串最后加一个\0 |

## 杂物

### 单调队列/栈的使用Tips

|  |
| --- |
| 单调栈用于构造笛卡尔树，笛卡尔树和平衡树也有一些联系。  这方面代码想想就会写  单调队列用于应付一些DP  斜率优化单调队列的push和pop主要在于叉积的正负和状态的优劣，想好不难写。 |

### 高斯消元 同余模方程/异或方程组

|  |
| --- |
| 备不时之需  void work()  {    int i , j , r , c , x , sum;    memset(a , 0 , sizeof(a));    for (i = 1 ; i <= m ;i ++)    {      scanf("%d %s %s",&j , ss , tt);      a[i][n + 1] = (week[tt] - week[ss] + 8) % 7;      while (j --)        scanf("%d",&x) , ++ a[i][x];      for (j = 1 ; j <= n ;j ++) a[i][j] %= 7;    }    ++ n;    for (r = 1 ,c = 1 ; r <= m && c <= n ;++ r ,++ c)    {      for (i = r ; i <= m ;++ i)        if (a[i][c]) break;      if (i > m)  {-- r;continue;}      if (i != r) for (j = 1 ; j <= n ;++ j) swap(a[r][j] , a[i][j]);      x = inv[a[r][c]]; if (x != 1)      for (j = c ; j <= n ;j ++)        a[r][j] \*= x , a[r][j] %= 7;      for (i = 1 ; i <= m ; i ++) if (i != r)      {        x = a[i][c]; if (x)        for (j = c ; j <= n ;j ++)          a[i][j] -= (x \* a[r][j]) % 7 , a[i][j] = (a[i][j] + 7) % 7;      }    }    sum = r - 1;    if (a[sum][n - 1] == 0 && a[sum][n] != 0)      puts("Inconsistent data.");    else if (sum < n - 1)      puts("Multiple solutions.");    else      for (i = 1 ; i < n ;i ++)        printf("%d%c",ans[a[i][n]] , i == n - 1 ? '\n' :' ');  } |

### 多项式乘法替代矩阵求线性递推数列

|  |
| --- |
| LL n , Q = 1e9 + 7 , c[N];  int m , a[N] , K , b[N] , M;  LL d[N];  bool vis[N];  struct poly  {    int a[N];    poly() {memset(a , 0 , sizeof(a));}  }f[N];  poly operator \* (poly x , poly y)  {    int i , j;    poly ans;    for (i = 0 ; i < M ; ++ i) if (x.a[i])      for (j = 0 ; j < M ; ++ j) if (y.a[j])      {        ans.a[i + j] += (LL)x.a[i] \* y.a[j] % Q;        if (ans.a[i + j] >= Q) ans.a[i + j] -= Q;      }    for (i = M ; i < M + M - 1 ; ++ i)      for (j = 0 ; j < M ; ++ j)      {        ans.a[j] += (LL)ans.a[i] \* f[i].a[j] % Q;        if (ans.a[j] >= Q) ans.a[j] -= Q;      }    return ans;  }  poly cal(LL k)  {    if (k < M + M - 1)      return f[k];    poly tmp = cal(k >> 1);    if (k & 1)      return tmp \* tmp \* f[1];    else return tmp \* tmp;  }  void work()  {    int i , j , k , x;    scanf("%d",&m);    for (i = 1 ; i <= m ; ++ i)      scanf("%d",&a[i]);    sort(a + 1 , a + m + 1);    memset(vis , 0 , sizeof(vis)) , vis[0] = 1;    scanf("%d",&K);    for (i = 1 ; i <= K ; ++ i)      scanf("%d",&b[i]) , vis[b[i]] = 1;    sort(b + 1 , b + 1 + K);    M = b[K];    memset(c , 0 , sizeof(c));    memset(d , 0 , sizeof(d));    memset(f , 0 , sizeof(f));    c[0] = d[0] = 1;    for (i = 1 ; i <= M ; ++ i)      for (j = 1 ; j <= m ; ++ j)        if (i - a[j] >= 0)          c[i] += c[i - a[j]] , c[i] %= Q;    if (M == 1)    {      printf("1\n");      return;    }    for (i = 0 ; i < M ; ++ i)    {      f[i].a[i] = 1;      for (j = 1 ; j <= K && b[j] <= i ; ++ j)      {        d[i] += c[b[j]] \* d[i - b[j]] % Q;        if (d[i] >= Q) d[i] -= Q;      }    }    for (j = 1 ; j <= K && b[j] <= M ; ++ j)      f[M].a[M - b[j]] += c[b[j]];    for (i = M + 1 ; i < M + M - 1 ; ++ i)    {      for (j = 0 ; j < M ; ++ j)      {        if (j) f[i].a[j] = f[i - 1].a[j - 1];        f[i].a[j] += (LL)f[M].a[j] \* f[i - 1].a[M - 1] % Q;        if (f[i].a[j] >= Q) f[i].a[j] -= Q;      }    }    poly ans = cal(n);    int ALL = 0;    for (i = 0 ; i < M ; ++ i)      ALL += d[i] \* ans.a[i] % Q , ALL %= Q;    printf("%d\n" , ALL);  } |

### 树的点分治

|  |
| --- |
| //树上两点间距离不大于K的对数  //关键在于重心的求法以及统计时要去掉一些。  #define N 100005  int n , L , W , pre[N] , mcnt;  struct edge  {    int x , w , next;  }e[N << 1];  bool f[N];  int s[N];  long long ans ;  pair<int , int> Find(int x , int fa , int cnt)  {    s[x] = 1; int m = 0;    pair<int , int> res = make\_pair(1 << 30 , -1);    for (int i = pre[x] ; ~i ; i = e[i].next)      if (!f[e[i].x] && e[i].x != fa)      {        res = min(res , Find(e[i].x , x , cnt));        m = max(s[e[i].x] , m);        s[x] += s[e[i].x];      }    m = max(m , cnt - s[x]);    return res = min(res , make\_pair(m , x));  }  pair<int , int> ds[N] , tds[N];  int s1 , s2;  void Getdis(int x , int fa , int d1 , int d2)  {    tds[s2 ++] = make\_pair(d2 , d1);    for (int i = pre[x] ; ~i ; i = e[i].next)      if (!f[e[i].x] && e[i].x != fa)        Getdis(e[i].x , x , d1 + 1 , d2 + e[i].w);  }  int C[N];  void add(int x , int w)  {    for (int i = x ; i <= L + 1 ; i += i & -i)      C[i] += w;  }  int sum(int x)  {    int res = 0;    for (int i = x ; i ; i -= i & -i)      res += C[i];    return res;  }  long long count(pair<int , int>\* P , int num)  {    long long res = 0;    sort(P , P + num);    int i , j;    for (int i = 0 ; i < num ; ++ i)      add(P[i].se + 1 , 1) , res -= (P[i].se \* 2 <= L && P[i].fi \* 2 <= W);    for (i = 0 , j = num - 1 ; i < num ; ++ i)    {      while (j >= 0 && P[i].fi + P[j].fi > W)        add(P[j].se + 1 , -1) , -- j;      if (L - P[i].se >= 0)        res += sum(L - P[i].se + 1);    }    while (j >= 0)      add(P[j].se + 1 , -1) , -- j;    return res >> 1;  }  void divide(int x , int cnt)  {    int i;    s1 = 1 , ds[0] = (make\_pair(0 , 0));    x = Find(x , 0 , cnt).se;    f[x] = 1;    for (i = pre[x] ; ~i ; i = e[i].next)      if (!f[e[i].x])      {        Getdis(e[i].x , x , 1 , e[i].w);        s[e[i].x] = s2;        ans -= count(tds , s2);        while (s2)          ds[s1 ++] = tds[-- s2];        //ds.insert(ds.end() , tds.begin() ,  tds.end());      }    ans += count(ds , s1);    for (i = pre[x] ; ~i ; i = e[i].next)      if (!f[e[i].x])        divide(e[i].x , s[e[i].x]);  }  void work()  {    int i , x , y , z;    scanf("%d%d%d",&n,&L,&W);    memset(pre , -1 , sizeof(pre)) , mcnt = 0;    for (i = 1 ; i < n ; ++ i)    {      scanf("%d%d",&x,&z);      e[mcnt] = (edge) {i + 1 , z , pre[x]} , pre[x] = mcnt ++;      e[mcnt] = (edge) {x , z , pre[i + 1]} , pre[i + 1] = mcnt ++;    }    memset(f , 0 , sizeof(f));    ans = 0;    divide(1 , n);    printf("%I64d\n" , ans);  }  int main()  {    freopen("~input.txt" , "r" , stdin);    work();    return 0;  } |

### 斯坦纳树

|  |
| --- |
| 连通集合中一些点的最小边权代价，此代码为前K个点与后K的点  int n , m , pre[55] , mcnt , K;  struct edge  {    int x , w , next;  }e[2005];  int dp[55][1 << 10] , id[55] , d[1 << 10];  bool f[55][1 << 10];  queue< pair<int , int> > Q;  void SPFA()  {    int x , w , i;    while (!Q.empty())    {      x = Q.front().fi , w = Q.front().se , Q.pop() , f[x][w] = 0;      for (i = pre[x] ; ~i ; i = e[i].next)      {        int y = e[i].x;        if (dp[x][w] + e[i].w < dp[y][w | id[y]])        {          dp[y][w | id[y]] = dp[x][w] + e[i].w;          if (!f[y][w | id[y]])            f[y][w | id[y]] = 1 , Q.push(make\_pair(y , w | id[y]));        }      }    }  }  bool check(int x)  {    int sum = 0;    for (int i = 0 ; i < K + K ; ++ i)      if (x & (1 << i))      {        if (i < K) -- sum;        else ++ sum;      }    return sum == 0;  }  void work()  {    int i , j , x , y , z;    scanf("%d%d%d",&n,&m,&K);    memset(pre , -1 , sizeof(pre));    mcnt = 0;    for (i = 1 ; i <= m ; ++ i)    {      scanf("%d%d%d",&x,&y,&z);      e[mcnt] = (edge){y , z , pre[x]} , pre[x] = mcnt ++;      e[mcnt] = (edge){x , z , pre[y]} , pre[y] = mcnt ++;    }    memset(id , 0 , sizeof(id));    memset(f , 0 , sizeof(f));    for (i = 1 ; i <= n ; ++ i)      for (j = 0 ; j < 1 << K + K ; ++ j)        dp[i][j] = 1 << 29;  **for (i = 1 ; i <= K ; ++ i)**  **{**  **id[i] = 1 << i - 1 , dp[i][id[i]] = 0;**  **id[n - i + 1] = 1 << K + i - 1 , dp[n - i + 1][id[n - i + 1]] = 0;**  **}//改在这里**    for (i = 0 ; i < 1 << K + K ; ++ i)    {      for (x = 1 ; x <= n ; ++ x)      {        for(j = (i - 1) & i ; j ; j = (j - 1) & i)          dp[x][i] = min(dp[x][i] , dp[x][j | id[x]] + dp[x][(i - j) | id[x]]);        if (dp[x][i] < 1 << 29) Q.push(make\_pair(x , i)) , f[x][i] = 1;      }      SPFA();    }    for (i = 0 ; i < 1 << K + K ; ++ i) if (check(i))    {      d[i] = 1 << 30;      for (j = 1 ; j <= n ; ++ j)        d[i] = min(d[i] , dp[j][i]);      for(j = (i - 1) & i ; j ; j = (j - 1) & i)        if (check(j))          d[i] = min(d[i] , d[j] + d[i - j]);    }    n = (1 << K + K) - 1;    if (d[n] >= 1 << 29)      puts("No solution");    else printf("%d\n" , d[n]);  }  int main()  {    freopen("~input.txt" , "r" , stdin);    //while (scanf("%d",&n) , n)    int \_; scanf("%d",&\_); while (\_--)      work();    return 0;  } |

### 矩形面积并

|  |
| --- |
| int n , m , d[N] , D , cnt , ca;  int price[5];  int rect[N >> 1][4] , type[N >> 1];  int **id**(int l , int r) {**return** l + r | l != r;}  #define MID int mid = (l + r) >> 1;  #define ID **id**(l , r)  #define Left l , mid  #define Right mid + 1 , r  **struct** stree  {  int ms , mn , add;  }t[N << 1];  void **pushup**(int l , int r)  {  MID; int p = ID , LL = **id**(Left) , RR = **id**(Right);  t[p].mn = t[LL].mn , t[p].ms = t[LL].ms;  **if** (t[RR].mn < t[p].mn)  t[p].mn = t[RR].mn , t[p].ms = 0;  **if** (t[RR].mn == t[p].mn) t[p].ms += t[RR].ms;  }  void **update**(int p , int val)  {  t[p].add += val;  t[p].mn += val;  }  void **pushdown**(int l , int r)  {  MID; int p = ID , LL = **id**(Left) , RR = **id**(Right);  **if** (t[p].add != 0)  {  **update**(LL , t[p].add) , **update**(RR , t[p].add);  t[p].add = 0;  }  }  void **update**(int l , int r , int top , int bot , int w)  {  **if** (top <= l && r <= bot) {int p = ID;  **update**(p , w);  **return**;}MID;**pushdown**(l , r);  **if** (top <= mid) **update**(Left , top , bot , w);  **if** (bot > mid) **update**(Right , top , bot , w);  **pushup**(l , r);  }  void **Build**(int l , int r) *// d[l] ~ d[r + 1];*  {  t[ID].add = 0;  **if** (l == r) { int p = ID;  t[p].ms = d[r] - d[l - 1];  t[p].mn = t[p].add = 0;  *//printf("%d : %d , %d : %d \n" , r , d[l] , d[r + 1] , t[ID].ms );*  **return**;}  MID;**Build**(Left);**Build**(Right);**pushup**(l , r);  }  **struct** seg  {  int x , y1 , y2 , w;  bool **operator** < (**const** seg& r) **const**{  **return** **make\_pair**(x , w) < **make\_pair**(r.x , r.w);}  }S[N];  void **work**()//代码实际上是体积并，但高度只有三种。  {  int i , j , k , x , W;  **scanf**("%d%d",&n,&m);  **printf**("Case %d: " , ++ ca);  **for** (i = 1 ; i <= m ; ++ i)  **scanf**("%d",&price[i]);  D = 0;  **for** (i = 1 ; i <= n ; ++ i)  {  **for** (j = 0 ; j < 4 ; ++ j)  **scanf**("%d",&rect[i][j]);  **scanf**("%d",&x) , type[i] = price[x];  d[D ++] = rect[i][1] , d[D ++] = rect[i][3];  }  **sort**(d , d + D);  D = **unique**(d , d + D) - d;  W = d[D - 1] - d[0];  **sort**(price + 1 , price + 1 + m);  LL ans = 0 , sum = 0;  **for** (k = 1 ; k <= m ; ++ k)  {  sum = cnt = 0;  **for** (i = 1 ; i <= n ; ++ i)  **if** (type[i] >= price[k])  {  int l = **lower\_bound**(d , d + D , rect[i][1]) - d + 1;  int r = **lower\_bound**(d , d + D , rect[i][3]) - d;  S[cnt ++] = (seg) {rect[i][0] , l , r , 1};  S[cnt ++] = (seg) {rect[i][2] , l , r , -1};  }  **sort**(S , S + cnt);  **Build**(1 , D - 1);  **for** (i = 0 ; i < cnt ; ++ i)  {  **if** (i && S[i].x > S[i - 1].x)  {  **if** (t[**id**(1 , D - 1)].mn != 0)  sum += (LL)W \* (S[i].x - S[i - 1].x);  **else** sum += (LL)(W - t[**id**(1 , D - 1)].ms) \* (S[i].x - S[i - 1].x);  }  **update**(1 , D - 1 , S[i].y1 , S[i].y2 , S[i].w);  }  ans += sum \* (price[k] - price[k - 1]);  }  cout << ans << endl;  } |