**汕头大学2016年**

**ACM算法模版**

代码贡献：黎炜烨，林翰宇，余楚放，谭达强，王滨

审校：林翰宇

## 图论

代码贡献:林翰宇

并查集 3-5

SPFA最短路 5-7

Prim最小生成树 7-8

Kruscal最小生成树 8-9

ISAP最大流 9-12

SPFA最小费用流（最大流）12-14

SPFA消负圈算法 15－16

迪杰斯特最小费用流16-18

迪杰斯特最短路18

Floyed最短路 19

### 并查集

**初始化：**

#include <memory.h>

const int NUM;

int UFSet[NUM];

void init(){

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

}

**Find：**

int Find(int root){

if (UFSet[root] <= 0){

return root;

} else{

return UFSet[root] = Find(UFSet[root]);

}

}

**Union By Rank**

void Merge(int root1,int root2){

root1 = Find(root1);

root2 = Find(root2);

if (root1 == root2){

return;

}

if (UFSet[root1] < UFSet[root2]){

//阶为负数.值小则绝对值大,阶大

UFSet[root2] = root1;

} else{

if (UFSet[root1] == UFSet[root2]){

UFSet[root2]--;

}

UFSet[root1] = root2;

}

}

### Union By Count,此时小于0的节点的绝对值，对应了该树节点数:

void Merge(int rt1,int rt2){

rt1 = Find(rt1);

rt2 = Find(rt2);

if (rt1 == rt2) return;

if (UFSet[rt1] < UFSet[rt2]){

UFSet[rt1] += UFSet[rt2];

UFSet[rt2] = rt1;

} else{

UFSet[rt2] += UFSet[rt1];

UFSet[rt1] = rt2;

}

}

### 给出并查集集森林节点最多的树的节点数(基于Union By Count)：

int countNode(int left,int right){

counter.clear();

int Max=0;

for (int i = left; i <= right ; ++i) {

if (UFSet[i] < 0 && Max < -UFSet[i])

Max = -UFSet[i];

}

}

return Max;

}

### 相离并查基：

给一个offset，各个offset后的值代表“可能性”。如a，b分属A，B，则（a＋offset，b）（a，b＋offset），若出现a，b根相同，则为矛盾情况。

### 删点并查集：

维护一个cur。删点时，令被删点指向cur，令被删点的根的值+1,cur++

### SPFA最短路

### 初始化：

const int MAXE;

const int MAXV;

struct node{

int to,next,cost;

}G[MAXE];

int cur;

int negV;

int head[MAXV];

int preve[MAXV];

//记录弧段

void add\_Node(int u,int v,int cost){

G[cur].cap = cap;

G[cur].to = v;

G[cur].next = head[u];

head[u] = cur ++;

}

void init(){

cur = 0;

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

}

### 带判负环，路径记录的spfa：

int spfa(int begin,int end,int n){

static bool outqueue[MAXV];

static int in[MAXV];

memset(outqueue,1, sizeof(outqueue));

// fill(dis,dis+MAXV,INF);

memset(dis,0x3F, sizeof(dis));

memset(in,0, sizeof(in));

queue<int > Q;

Q.push(begin);

dis[begin] = 0;

in[begin] = 1;

while (!Q.empty()){

int s = Q.front();

Q.pop();

outqueue[s] = true;

for (int i = head[s]; ~i ; i = G[i].next) {

if (dis[G[i].to] > dis[s] + G[i].cost){

dis[G[i].to] = dis[s] + G[i].cost;

preve[G[i].to] = i;

if (++in[G[i].to] >= n)

{

negV = G[i].to;

return -1;

}

if (outqueue[G[i].to]){

outqueue[G[i].to] = false;

Q.push(G[i].to);

}

}

}

}

return dis[end];

}

SPFA最长路：

改变正负号

差分约束：给一超级源点s，连接其他所有点。若不存在负环，则方程有解。解为f(t)-f(s)

### Prim最小生成树

### 初始化：

#include<memory.h>

const int NUM;

const int INF = 0x7FFFFFFF;

int rls[NUM][NUM],n;

### 最小生成树：

int min\_span\_tree(int begin){

static int low\_cost[NUM];

static bool in\_tree[NUM];

int Min,count=0,Min\_sub;

bool flag;

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

low\_cost[i] = rls[begin][i];

}

memset(in\_tree,0, sizeof(bool)\*n);

in\_tree[begin] = true;

for (int i = 0; i < n-1; ++i) {

Min = INF;

flag = true;

for (int j = 0; j < n; ++j) {

if (!in\_tree[j] && low\_cost[j]<Min) {

Min = low\_cost[j];

Min\_sub = j;

flag = false;

}

}

if (flag) return -1;

count += Min;

in\_tree[Min\_sub] = true;

for (int j = 0; j < n; ++j) {

if (!in\_tree[j] && Min\_sub!=j && rls[Min\_sub][j] < low\_cost[j]) {

low\_cost[j] = rls[Min\_sub][j];

}

}

}

return count;

}

### 最大生成树：

改变正负号

### Kruscal最小生成树

### 初始化：

并查集Union By Rank

struct edge{

int u,v,weight;

edge(){}

edge(int u,int v,int w):u(u),v(v),weight(w){}

bool operator<(const edge & e2) const{

return weight > e2.weight;

}

};

priority\_queue<edge> Q;

### Kruscal:

int Kruscal(priority\_queue<edge> & Q){

init();

int res = 0;

while(!Q.empty()){

edge e = Q.top();

Q.pop();

if (Find(e.u) != Find(e.v)){

Merge(e.u,e.v);

res+=e.weight;

}

}

return res;

}

### ISAP最大流:

### 初始化：

#include <queue>

#include<memory.h>

#include <algorithm>

using namespace std;

const int MAXE;

const int MAXV;;

const int INF = 0x7FFFFFFF;

int dst;

int src;

int cur;

struct Node{

int cap,to,next;

}G[MAXE];

int level[MAXV];

int lAppear[MAXV];

int head[MAXV];

int Gcur[MAXV];

int pre[MAXV];

void init(){

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

cur = 0;

src =;

dst =;

}

void add\_Edge(int u,int v,int cap){

G[cur].to = v;

G[cur].cap = cap;

G[cur].next = head[u];

head[u] = cur++;

}

void addEdge(int u,int v,int cap){

add\_Edge(u,v,cap);

add\_Edge(v,u,0);

}

### ISAP:

void refBfs(int u){

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

memset(lAppear,0, sizeof(lAppear));

queue<int> Q;

Q.push(u);

level[u] = 0;

while (!Q.empty()){

int s = Q.front();

Q.pop();

for (int i = head[s]; ~i; i = G[i].next) {

if (level[G[i].to] < 0){

level[G[i].to] = level[s] + 1;

lAppear[level[G[i].to]]++;

Q.push(G[i].to);

}

}

}

}

int maxFlow(int s,int t,int n){

//n为最长可能路径

memcpy(Gcur,head, sizeof(Gcur));

refBfs(t);

int flow = 0,u;

pre[s] = u = s;

while (level[s] < n){

if (u == t){

int neck = u,f=INF;

for (int i = s; i!=t ; i = G[Gcur[i]].to) {

if (f > G[Gcur[i]].cap){

f = G[Gcur[i]].cap;

neck = i;

}

}

for (int i = s; i != t ; i = G[Gcur[i]].to) {

G[Gcur[i]].cap -= f;

G[Gcur[i]^1].cap += f;

}

flow += f;

u = neck;

}

int i;

for (i = Gcur[u]; ~i ; i = G[i].next) {

if (G[i].cap && level[G[i].to] + 1 == level[u]) break;

}

if (~i){

Gcur[u] = i;

pre[G[i].to] = u;

u = G[i].to;

} else{

if (--lAppear[level[u]] == 0) break;

int mind = n;

for (int i = head[u]; ~i ; i = G[i].next) {

if (G[i].cap && level[G[i].to] < mind){

mind = level[G[i].to];

Gcur[u] = i;

}

}

level[u] = mind + 1;

lAppear[level[u]]++;

u = pre[u];

}

}

return flow;

}

### SPFA最小费用流

### SPFA：参照模版。改queue为stack，松弛条件改为

if (G[i].cap && dis[G[i].to] > dis[s] + G[i].cost){

### 初始化：

#include <queue>

#include<memory.h>

#include <algorithm>

using namespace std;

const int MAXE;

const int MAXV;;

const int INF = 0x7FFFFFFF;

int dst;

int src;

int cur;

struct Node{

int cap,to,next,cost;

}G[MAXE];

int level[MAXV];

int lAppear[MAXV];

int head[MAXV];

int Gcur[MAXV];

int pre[MAXV];

void add\_Node(int u,int v,int cap,int cost){

G[cur].cap = cap;

G[cur].to = v;

G[cur].cost = cost;

G[cur].next = head[u];

head[u] = cur ++;

}

void addNode(int u,int v,int cap,int cost){

add\_Node(u,v,cap,cost);

add\_Node(v,u,0,-cost);

}

void init(){

src = ;

dst = ;

cur = 0;

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

}

### 最小费用流：

int mincost(int s,int t,int f){

int res = 0;

while (f > 0 && ~spfa(s,t,MAXV)){

int d = INF;

for (int i = t; i != s ; i = G[preve[i] ^ 1].to) {

d = min(d,G[preve[i]].cap);

}

f -= d;

res += dis[t] \* d;

for (int i = t; i != s ; i = G[preve[i] ^ 1].to) {

G[preve[i]].cap -= d;

G[preve[i] ^ 1].cap += d;

}

}

return res;

}

### 最小费用最大流：

int flow,cost;

void minCostMaxFlow(int s,int t){

cost = 0;

flow = 0;

while (~spfa(s,t,MAXV)){

int d = INF;

for (int i = t; i != s ; i = G[preve[i] ^ 1].to) {

d = min(d,G[preve[i]].cap);

}

flow += d;

cost += dis[t] \* d;

for (int i = t; i != s ; i = G[preve[i] ^ 1].to) {

G[preve[i]].cap -= d;

G[preve[i] ^ 1].cap += d;

}

}

}

### SPFA消负圈算法:

### 初始化：

同SPFA最小费用流

构建残余网络图，注意更新dst节点的流

scanf("%d",&tmp);

fj[j] += tmp;

add\_Node(i,j+n,INF,c);

add\_Node(j+n,i,tmp,-c);

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

add\_Node(i+n,dst,c[i]-fj[i],0);

add\_Node(dst,i+n,fj[i],0);

}

int k = spfa(src,dst,m+n+2);

//

if (~k){

puts("OPTIMAL");

} else{

augument();

}

### 负圈增广：

void augument(){

static bool vis[MAXV];

memset(vis,0, sizeof(vis));

int u;

for (u = negV; !vis[u] ; u = G[preve[u] ^ 1].to) {

vis[u] = true;

}

int f = INF;

bool flg = true;

for (int i = u; i != u || flg; i = G[preve[i] ^ 1].to) {

f = min(G[preve[i]].cap,f);

flg = false;

}

flg = true;

for (int i = u; i != u || flg ; i = G[preve[i] ^ 1].to) {

G[preve[i]].cap -= f;

G[preve[i] ^ 1].cap += f;

flg = false;

}

}

### 迪杰斯特最小费用流：

const int MAXN = 110;

const int MAXE = 30500;

const int MAXV = 300;

const int INF = 0x6FFFFFFF;

//

int src,dst,n,m;

//int x[MAXN],y[MAXN],b[MAXN],q[MAXN],c[MAXN],p[MAXN];

struct edge{

int to,cap,cost,rev;

edge(){}

edge(int t,int c,int cost,int rev):to(t),cap(c),cost(cost),rev(rev){}

};

typedef pair<int,int> P;

int V;

vector<edge> G[MAXV];

int h[MAXV];

int dist[MAXV];

int src,dst;

int prevv[MAXV],preve[MAXV];

//int GM[MAXV][MAXV];

void addNode(int u,int v,int cap,int cost){

G[u].push\_back(edge(v,cap,cost,G[v].size()));

G[v].push\_back(edge(u,0,-cost,G[u].size() - 1));

}

int MinCostFlow(int s,int t,int f){

int res=0;

fill(h,h+V,0);

while(f>0){

priority\_queue<P,vector<P>,greater<P> > Q;

fill(dist,dist+V,INF);

dist[s] = 0;

Q.push(P(0,s));

while(!Q.empty()){

P p = Q.top();

Q.pop();

int v = p.second;

if (dist[v] < p.first) continue;

for (int i = 0; i < G[v].size(); ++i) {

edge & e = G[v][i];

if (e.cap > 0 && dist[e.to] > dist[v] + e.cost + h[v] - h[e.to]){

dist[e.to] = dist[v] + e.cost + h[v] - h[e.to];

prevv[e.to] = v;

preve[e.to] = i;

Q.push(P(dist[e.to],e.to));

}

}

}

if (dist[t] == INF){

return -1;

}

for (int i = 0; i < V; ++i) {

h[i] += dist[i];

}

int d = f;

for (int v = t; v !=s ; v = prevv[v]) {

d = min(d,G[prevv[v]][preve[v]].cap);

}

f -= d;

res += d \* h[t];

for (int v = t; v != s ; v = prevv[v]) {

edge & e = G[prevv[v]][preve[v]];

e.cap -= d;

G[e.to][e.rev].cap += d;

}

}

return res;

}

### 迪杰斯特最短路

### 初始化：参照迪杰斯特最小费用流，改用手工临接表实现（SPFA）

priority\_queue<P,vector<P>,greater<P> > Q;

fill(dist,dist+V,INF);

dist[s] = 0;

Q.push(P(0,s));

while(!Q.empty()){

P p = Q.top();

Q.pop();

int u = p.second;

if (dist[u] < p.first) continue;

for (int i = head[u]; ~i ; i = G[u].next) {

edge & e = G[i];

if (dist[e.to] > dist[v] + e.cost){

dist[e.to] = dist[v] + e.cost];

prevv[e.to] = v;

preve[e.to] = i;

Q.push(P(dist[e.to],e.to));

}

}

### Floyed最短路：

### 初始化：

memset(dp,0x3F,sizeof(dp));

dp[i][i] = 0;

dp[i]j[j] = map[i][j]

for (int i = 1; i <= upbound;++i) {

for (int j = 1; j <= upbound; ++j) {

for (int k = 1; k <= upbound; ++k) {

dp[j][k] = min(dp[j][i] + dp[i][k],dp[j][k]);

}

}

}

### 二分匹配：

const int MAXV;

int V;

vector<int > G[MAXV];

int match[MAXV];

bool used[MAXV];

void addAdge(int u,int v){

G[u].push\_back(v);

G[v].push\_back(u);

}

bool dfs(int u){

used[u] = true;

for (int i = head[u]; ~i ; i = Buffer[i].next) {

int v = Buffer[i].to;

int w = match[v];

if (w < 0 || !used[w] && dfs(w)){

match[u] = v;

match[v] = u;

return true;

}

}

return false;

}

void bipartiteMatchJudge(int begin,int end,int Maxmatch){

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

int res=0;

for (int i = begin; i <= end; ++i) {

memset(used,0, sizeof(used));

if (dfs(i)) {

if (++res == Maxmatch) {

printf("%d\n",i/d + 1);

return;

}

}

}

puts("impossible");

}

## 1.2线段树

代码贡献：林翰宇

### 撸上撸下:

PushUp:sum[rt] = f(sum[L(rt)],sum[R(rt)]);

PushDown:if(col[rt]) …

### 一般构建：

void build(int l,int r,int rt) {

if (l == r) {

return ;

}

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

build(lson);

build(rson);

PushUP(rt);

}

### 区间查询：

int query(int L,int R,int l,int r,int rt) {

if (L <= l && r <= R) {

return sum[rt];

}

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

int ret = 0;

if (L <= m) ret += query(L , R , lson);

if (R > m) ret += query(L , R , rson);

return ret;

}

int main() {

### 单点更新：

void update(int p,int add,int l,int r,int rt) {

if (l == r) {

sum[rt] += add;

return ;

}

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

if (p <= m) update(p , add , lson);

else update(p , add , rson);

PushUP(rt);

}

### 段更新：

void update(int L,int R,int c,int l,int r,int rt) {

if (L <= l && r <= R) {

return ;

}

PushDown(rt);

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

if (L <= m) update(L , R , c , lson);

if (R > m) update(L , R , c , rson);

PushUp(rt);

}

### 重要例题：

* + [hdu1542 Atlantis](http://acm.hdu.edu.cn/showproblem.php?pid=1542)  
    题意:矩形面积并  
    思路:浮点数先要离散化;然后把矩形分成两条边,上边和下边,对横轴建树,然后从下到上扫描上去,用cnt表示该区间下边比上边多几个  
    线段树操作:update:区间增减 query:直接取根节点的值

[**?**](http://www.ericbess.com/ericblog/2008/03/03/wp-codebox/#examples)[**View Code**](javascript:;)**CPP**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78 | #include <cstdio>  #include <cstring>  #include <cctype>  #include <algorithm>  using namespace std;  #define lson l , m , rt << 1  #define rson m + 1 , r , rt << 1 | 1    const int maxn = 2222;  int cnt[maxn << 2];  double sum[maxn << 2];  double X[maxn];  struct Seg {  double h , l , r;  int s;  Seg(){}  Seg(double a,double b,double c,int d) : l(a) , r(b) , h(c) , s(d) {}  bool operator < (const Seg &cmp) const {  return h < cmp.h;  }  }ss[maxn];  void PushUp(int rt,int l,int r) {  if (cnt[rt]) sum[rt] = X[r+1] - X[l];  else if (l == r) sum[rt] = 0;  else sum[rt] = sum[rt<<1] + sum[rt<<1|1];  }  void update(int L,int R,int c,int l,int r,int rt) {  if (L <= l && r <= R) {  cnt[rt] += c;  PushUp(rt , l , r);  return ;  }  int m = (l + r) >> 1;  if (L <= m) update(L , R , c , lson);  if (m < R) update(L , R , c , rson);  PushUp(rt , l , r);  }  int Bin(double key,int n,double X[]) {  int l = 0 , r = n - 1;  while (l <= r) {  int m = (l + r) >> 1;  if (X[m] == key) return m;  if (X[m] < key) l = m + 1;  else r = m - 1;  }  return -1;  }  int main() {  int n , cas = 1;  while (~scanf("%d",&n) && n) {  int m = 0;  while (n --) {  double a , b , c , d;  scanf("%lf%lf%lf%lf",&a,&b,&c,&d);  X[m] = a;  ss[m++] = Seg(a , c , b , 1);  X[m] = c;  ss[m++] = Seg(a , c , d , -1);  }  sort(X , X + m);  sort(ss , ss + m);  int k = 1;  for (int i = 1 ; i < m ; i ++) {  if (X[i] != X[i-1]) X[k++] = X[i];  }  memset(cnt , 0 , sizeof(cnt));  memset(sum , 0 , sizeof(sum));  double ret = 0;  for (int i = 0 ; i < m - 1 ; i ++) {  int l = Bin(ss[i].l , k , X);  int r = Bin(ss[i].r , k , X) - 1;  if (l <= r) update(l , r , ss[i].s , 0 , k - 1, 1);  ret += sum[1] \* (ss[i+1].h - ss[i].h);  }  printf("Test case #%d**\n**Total explored area: %.2lf**\n\n**",cas++ , ret);  }  return 0;  } |

* + [hdu1828 Picture](http://acm.hdu.edu.cn/showproblem.php?pid=1828)  
    题意:矩形周长并  
    思路:与面积不同的地方是还要记录竖的边有几个(numseg记录),并且当边界重合的时候需要合并(用lbd和rbd表示边界来辅助)  
    线段树操作:update:区间增减 query:直接取根节点的值

[**?**](http://www.ericbess.com/ericblog/2008/03/03/wp-codebox/#examples)[**View Code**](javascript:;)**CPP**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73 | #include <cstdio>  #include <cstring>  #include <cctype>  #include <algorithm>  using namespace std;  #define lson l , m , rt << 1  #define rson m + 1 , r , rt << 1 | 1    const int maxn = 22222;  struct Seg{  int l , r , h , s;  Seg() {}  Seg(int a,int b,int c,int d):l(a) , r(b) , h(c) , s(d) {}  bool operator < (const Seg &cmp) const {  if (h == cmp.h) return s > cmp.s;  return h < cmp.h;  }  }ss[maxn];  bool lbd[maxn<<2] , rbd[maxn<<2];  int numseg[maxn<<2];  int cnt[maxn<<2];  int len[maxn<<2];  void PushUP(int rt,int l,int r) {  if (cnt[rt]) {  lbd[rt] = rbd[rt] = 1;  len[rt] = r - l + 1;  numseg[rt] = 2;  } else if (l == r) {  len[rt] = numseg[rt] = lbd[rt] = rbd[rt] = 0;  } else {  lbd[rt] = lbd[rt<<1];  rbd[rt] = rbd[rt<<1|1];  len[rt] = len[rt<<1] + len[rt<<1|1];  numseg[rt] = numseg[rt<<1] + numseg[rt<<1|1];  if (lbd[rt<<1|1] && rbd[rt<<1]) numseg[rt] -= 2;//两条线重合  }  }  void update(int L,int R,int c,int l,int r,int rt) {  if (L <= l && r <= R) {  cnt[rt] += c;  PushUP(rt , l , r);  return ;  }  int m = (l + r) >> 1;  if (L <= m) update(L , R , c , lson);  if (m < R) update(L , R , c , rson);  PushUP(rt , l , r);  }  int main() {  int n;  while (~scanf("%d",&n)) {  int m = 0;  int lbd = 10000, rbd = -10000;  for (int i = 0 ; i < n ; i ++) {  int a , b , c , d;  scanf("%d%d%d%d",&a,&b,&c,&d);  lbd = min(lbd , a);  rbd = max(rbd , c);  ss[m++] = Seg(a , c , b , 1);  ss[m++] = Seg(a , c , d , -1);  }  sort(ss , ss + m);  int ret = 0 , last = 0;  for (int i = 0 ; i < m ; i ++) {  if (ss[i].l < ss[i].r) update(ss[i].l , ss[i].r - 1 , ss[i].s , lbd , rbd - 1 , 1);  ret += numseg[1] \* (ss[i+1].h - ss[i].h);  ret += abs(len[1] - last);  last = len[1];  }  printf("%d**\n**",ret);  }  return 0;  } |

* 区间合并  
  这类题目会询问区间中满足条件的连续最长区间,所以PushUp的时候需要对左右儿子的区间进行合并
  + [poj3667 Hotel](http://poj.org/problem?id=3667)  
    题意:1 a:询问是不是有连续长度为a的空房间,有的话住进最左边  
    2 a b:将[a,a+b-1]的房间清空  
    思路:记录区间中最长的空房间  
    线段树操作:update:区间替换 query:询问满足条件的最左断点

[**?**](http://www.ericbess.com/ericblog/2008/03/03/wp-codebox/#examples)[**View Code**](javascript:;)**CPP**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77 | #include <cstdio>  #include <cstring>  #include <cctype>  #include <algorithm>  using namespace std;  #define lson l , m , rt << 1  #define rson m + 1 , r , rt << 1 | 1    const int maxn = 55555;  int lsum[maxn<<2] , rsum[maxn<<2] , msum[maxn<<2];  int cover[maxn<<2];    void PushDown(int rt,int m) {  if (cover[rt] != -1) {  cover[rt<<1] = cover[rt<<1|1] = cover[rt];  msum[rt<<1] = lsum[rt<<1] = rsum[rt<<1] = cover[rt] ? 0 : m - (m >> 1);  msum[rt<<1|1] = lsum[rt<<1|1] = rsum[rt<<1|1] = cover[rt] ? 0 : (m >> 1);  cover[rt] = -1;  }  }  void PushUp(int rt,int m) {  lsum[rt] = lsum[rt<<1];  rsum[rt] = rsum[rt<<1|1];  if (lsum[rt] == m - (m >> 1)) lsum[rt] += lsum[rt<<1|1];  if (rsum[rt] == (m >> 1)) rsum[rt] += rsum[rt<<1];  msum[rt] = max(lsum[rt<<1|1] + rsum[rt<<1] , max(msum[rt<<1] , msum[rt<<1|1]));  }  void build(int l,int r,int rt) {  msum[rt] = lsum[rt] = rsum[rt] = r - l + 1;  cover[rt] = -1;  if (l == r) return ;  int m = (l + r) >> 1;  build(lson);  build(rson);  }  void update(int L,int R,int c,int l,int r,int rt) {  if (L <= l && r <= R) {  msum[rt] = lsum[rt] = rsum[rt] = c ? 0 : r - l + 1;  cover[rt] = c;  return ;  }  PushDown(rt , r - l + 1);  int m = (l + r) >> 1;  if (L <= m) update(L , R , c , lson);  if (m < R) update(L , R , c , rson);  PushUp(rt , r - l + 1);  }  int query(int w,int l,int r,int rt) {  if (l == r) return l;  PushDown(rt , r - l + 1);  int m = (l + r) >> 1;  if (msum[rt<<1] >= w) return query(w , lson);  else if (rsum[rt<<1] + lsum[rt<<1|1] >= w) return m - rsum[rt<<1] + 1;  return query(w , rson);  }  int main() {  int n , m;  scanf("%d%d",&n,&m);  build(1 , n , 1);  while (m --) {  int op , a , b;  scanf("%d",&op);  if (op == 1) {  scanf("%d",&a);  if (msum[1] < a) puts("0");  else {  int p = query(a , 1 , n , 1);  printf("%d**\n**",p);  update(p , p + a - 1 , 1 , 1 , n , 1);  }  } else {  scanf("%d%d",&a,&b);  update(a , a + b - 1 , 0 , 1 , n , 1);  }  }  return 0;  } |

## 1.3DP

代码贡献：林翰宇

### 最长子序列 hdu1081

int cal\_max\_subsequence(int n){

int dp=0,Max=-INF;

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

dp = (dp+t\_array[i] < 0?0:dp) + t\_array[i];

Max = max(Max,dp);

}

return Max;

}

### 环形最长子列：

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

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

dp[i] -= cost;

dp[i+n] = dp[i];

}

len = 0;

sum = 0;

ans = 0;

for (int i = 0; i<n\*2; ++i) {

sum += dp[i];

len++;

if (sum<0){

sum = 0;

len = 0;

}

if (len > ans){

ans = len;

}

if (len >= n){

break;

}

}

### 滚动数组LCS（最长回文子串） hdu1513

int dp[2][NUM-1];

int main(){

int n;

char seq[NUM];

while (scanf("%d",&n)!=EOF){

scanf("%s",seq+1);

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

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

}

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

for (int j = 1; j <= n; ++j) {

int x,y;

x=i&1;

y=!x;

if(seq[i] == seq[n-j+1]){

dp[x][j] = dp[y][j-1] + 1;

} else{

dp[x][j] = max(dp[y][j],dp[x][j-1]);

}

}

}

printf("%d\n",n-dp[n&1][n]);

}

}

### 混合背包 hdu1059

void mix\_pack(int cost,int weight,int amount,int room,int ar[]){

int tmp = cost \* amount;

if (tmp >= room){

complete\_pack(cost,weight,room,ar);

} else{

for (int i = 1 ; i < amount; i<<=1) {

zero\_one\_pack(cost\*i,weight\*i,room,ar);

amount -= i;

}

zero\_one\_pack(amount \* cost,amount \* weight ,room,ar);

}

}

### 最大全1矩形 hdu1505

int zeroOneMaxArea(bool (\*map)[NUM],int m,int n){

static Node stack[NUM+10];

static int l[NUM];

static int h[NUM];

memset(h,0, sizeof(h));

int ans = -1;

for (int i = 1; i <= m; ++i) {

int top = 0;

//i行

for (int j = 1; j <= n; ++j) {

//j列

if (map[i][j]){

//滚动数组.把第j列从上到下连续者相加

h[j]++;

} else{

h[j] = 0;

}

//将连续递增的矩形入栈.当出现矮边界时退栈,计算矩形面积

if(top && stack[top].h > h[j]){

//如果栈顶元素高度大于第i行j列的连续高度

while(top && stack[top].h > h[j]){

ans = max(ans,stack[top].h \* (j-stack[top].l));

//不包括j

top--;

}

stack[++top].h = h[j];

}else{

stack[++top].h = h[j];

stack[top].l = j;

if (top >=2 && stack[top-1].h == stack[top].h) stack[top].l = stack[top-1].l;

//数字一样,则并入同一矩形.把最大高度入栈

}

}

while(top)

{

ans=max(ans,stack[top].h\*(n+1 - stack[top].l));

top--;

//处理边界

}

}

return ans;

}

int main(){

int t,n,m;

bool map[NUM][NUM];

char tmp[3];

scanf("%d",&t);

while (t--){

scanf("%d %d",&n,&m);

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

for (int j = 1; j <= m ; j++) {

scanf("%s",tmp);

switch (tmp[0]){

case 'R':

map[i][j] = false;

break;

case 'F':

map[i][j] = true;

break;

default:;

}

}

}

printf("%d\n",zeroOneMaxArea(map,n,m)\*3);

}

}

### LCIS最长上升公共子序列

template <class T>

int LCIS\_1D(int dp[],T seq1[],T seq2[],int seqLen1,int seqLen2){

memset(dp,0, sizeof(int) \* (max(seqLen1,seqLen2)+1));

int Max;

for (int i = 1; i <= seqLen1 ; ++i) {

Max = 0;

for (int j = 1; j <= seqLen2; ++j) {

if(seq1[i] > seq2[j] && Max < dp[j]) Max = dp[j];

if (seq1[i] == seq2[j]) dp[j] = Max+1;

}

}

Max = 0;

for (int j = 1; j <= seqLen2 ; ++j) {

Max = max(Max,dp[j]);

}

return Max;

}

## 1.4附录：杂题模版

### O(n)第k小（第k大）

代码贡献：林翰宇

#include <algorithm>

#include <iostream>

#define CUTOFF 9

#define NUM 50

using namespace std;

template <class T>

void insert\_sort(T ar[],int size){

//升序

for (int i = 1,j; i < size; ++i) {

T tmp = ar[i];

for (j = i ; j > 0 && tmp < ar[j-1] ; j--)

ar[j] = ar[j-1];

ar[j] = tmp;

}

}

template <class T>

T median\_3(T ar[], int left,int right){

int mid = (left+right)>>1;

if(ar[left] > ar[mid]){

swap(ar[left],ar[mid]);

}

if(ar[mid] > ar[right]){

swap(ar[right],ar[mid]);

}

if(ar[left] > ar[mid]){

swap(ar[left],ar[mid]);

}

swap(ar[mid],ar[--right]);

return ar[right];

}

template <class T>

void fast\_select(T ar[], int left,int right,int k){

//第k小

if (right-left>CUTOFF){

int pivot = median\_3(ar,left,right);

int i=left,j=right-1;

for (;;) {

while(ar[++i] <pivot);

while(ar[--j] >pivot);

if (i<j){

swap(ar[i],ar[j]);

} else{

break;

}

}

swap(ar[right-1],ar[i]);

if (k<i){

fast\_select(ar,left,i-1,k);

} else if (k>i){

fast\_select(ar,i+1,right,k);

}

} else{

insert\_sort(ar+left,right-left+1);

}

}

void prt\_ar(int ar[],int size){

for (int i = 0; i < size; ++i) {

cout<<"\t"<<ar[i];

}

cout<<endl;

}

int main(){

int a[NUM];

for (int j = 0; j < NUM; ++j) {

for (int i = 0; i < NUM; ++i) {

a[i] = i%10;

}

fast\_select(a,0,NUM-1,j);

//第a[k-1]就是所求(第k小)

cout<<a[j]<<endl;

}

return 0;

}

### 二分查找改进的LIS算法（hdu 1025）

代码贡献：林翰宇

int Binary\_Search(int A[],int len,int key){

int mid,left = 1,right = len;

while(left <= right){

mid = MID(left,right);

if (key < A[mid]){

right = mid-1;

} else if(key > A[mid]){

left = mid+1;

} else{

break;

}

}

return left;

}

int main(){

int count = 1,n;

// printf("%d",n);

while(scanf("%d",&n)!=EOF){

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

int tmp;

scanf("%d", &tmp);

scanf("%d", &dp[tmp]);

}

int len = 1,sub;

helper[1] = dp[1];

for (int i = 2; i <= n; ++i) {

sub = Binary\_Search(helper,len,dp[i]);

helper[sub] = dp[i];

if (sub > len){

len++;

}

}

if (len == 1){

printf("Case %d:\nMy king, at most 1 road can be built.\n\n",count++);

} else{

printf("Case %d:\nMy king, at most %d roads can be built.\n\n",count++,len);

}

}

}