目录

1. 博弈论
2. Wythoff(黄金分割，两堆石头，可以一起拿一样的)

|  |
| --- |
| /\* |
| 有2堆石子。A B两个人轮流拿，A先拿。每次可以从一堆中取任意个或从 |
| 2堆中取相同数量的石子，但不可不取。拿到最后1颗石子的人获胜。假设 |
| A B都非常聪明，拿石子的过程中不会出现失误。给出2堆石子的数量，问 |
| 最后谁能赢得比赛。 |
| \*/ |
| int main() |
| { |
| int t, a, b, m, k; |
| scanf("%d", &t); |
| while (t--) |
| { |
| scanf("%d%d", &a, &b); |
| if (a > b) |
| { |
| a ^= b; |
| b ^= a; |
| a ^= b; |
| } |
| m = b - a; |
| k = (int)(m \* (1 + sqrt(5)) / 2.0); |
| //m = ? \* a |
| //k = m / ? |
| //?:黄金分割数 |
| //如果a == k，则为后手赢，否则先手赢（奇异局） |
| printf("%s\n", a == k ? "B" : "A"); |
| } |
| return 0; |
| } |

1. Bash(石头谁先拿到最后一个)

|  |
| --- |
| /\* |
| 有一堆石子共有N个。A B两个人轮流拿，A先拿。每次最少拿1颗， |
| 最多拿K颗，拿到最后1颗石子的人获胜。假设A B都非常聪明， |
| 拿石子的过程中不会出现失误。给出N和K，问最后谁能赢得比赛。 |
| \*/ |
| /\* bashgame \*/ |
| int bash(int N, int K) |
| { |
| if (N % (K + 1) == 0) return 2; |
| return 1; |
| } |

1. nim游戏，N堆石子，每次可全拿

|  |
| --- |
| /\*有N堆石子。A B两个人轮流拿，A先拿。每次只能从一堆中取若干个， |
| 可将一堆全取走，但不可不取，拿到最后1颗石子的人获胜。假设A |
| B都非常聪明，拿石子的过程中不会出现失误。给出N及每堆石子的数量， |
| 问最后谁能赢得比赛。\*/ |
| /\*Nim 游戏\*/ |
| int main(int argc, const char \* argv[]) |
| { |
| int N, stone, tag = 0; |
| scanf("%d", &N); |
| while (N--){ |
| scanf("%d", &stone); |
| tag ^= stone; |
| } |
| //tag为0则为后手赢，否则为先手赢 |
| printf("%c\n", tag == 0 ? 'B' : 'A'); |
| return 0; |
| } |

1. sg函数

|  |
| --- |
| const int MAX\_DIG = 64; |
| //f[]：可以取走的石子个数 |
| //sg[]:0~n的SG函数值 |
| //hash[]:mex{} |
| int f[MAX\_DIG]={1,3},sg[MAX\_DIG],hash[MAX\_DIG]; |
| int k;//k是f[]的有效长度 |
| void getSG(int n) |
| { |
| memset(sg,0,sizeof(sg)); |
| for(int i=1; i<=n; i++) { |
| memset(hash,0,sizeof(hash)); |
| for(int j=0; f[j]<=i && j < k; j++) //k是f[]的有效长度 |
| hash[sg[i-f[j]]]=1; |
| for(int j=0; ; j++) { //求mes{}中未出现的最小的非负整数 |
| if(hash[j]==0) { |
| sg[i]=j; |
| break;} |
| } |
| } |
| } |
| int main(){ |
| int n; |
| read(n); |
| k=2; |
| getSG(n); |
| for(int i=0;i<=n;i++){ |
| cout<<sg[i]<<” ”; |
| } |
| return 0; |
| } |

1. Fefjasl
2. 树
   * 1. 二叉树
3. 中序遍历+后序遍历建二叉树

|  |
| --- |
| // UVa548 Tree |
| // Rujia Liu |
| // 题意：给一棵点带权（权各不相同，都是正整数）二叉树的中序和后序遍历，找一个叶子使得它到根的路径上的权和最小。如果有多解，该叶子本身的权应尽量小 |
| // 算法：递归建树，然后DFS。注意，直接递归求结果也可以，但是先建树的方法不仅直观，而且更好调试 |
| // 因为各个结点的权值各不相同且都是正整数，直接用权值作为结点编号 |
| const int maxv = 10000 + 10; |
| int in\_order[maxv], post\_order[maxv], lch[maxv], rch[maxv]; |
| int n; |
| bool read\_list(int\* a) { |
| string line; |
| if(!getline(cin, line)) return false; |
| stringstream ss(line); |
| n = 0; |
| int x; |
| while(ss >> x) a[n++] = x; |
| return n > 0; |
| } |
| // 把in\_order[L1..R1]和post\_order[L2..R2]建成一棵二叉树，返回树根 |
| int build(int L1, int R1, int L2, int R2) { |
| if(L1 > R1) return 0; // 空树 |
| int root = post\_order[R2]; |
| int p = L1; |
| while(in\_order[p] != root) p++; |
| int cnt = p-L1; // 左子树的结点个数 |
| lch[root] = build(L1, p-1, L2, L2+cnt-1); |
| rch[root] = build(p+1, R1, L2+cnt, R2-1); |
| return root;} |
| int best, best\_sum; // 目前为止的最优解和对应的权和 |
| void dfs(int u, int sum) { |
| sum += u; |
| if(!lch[u] && !rch[u]) // 叶子 |
| if(sum < best\_sum || (sum == best\_sum && u < best)) { best = u; best\_sum = sum; } |
| if(lch[u]) dfs(lch[u], sum); |
| if(rch[u]) dfs(rch[u], sum); |
| } |
| int main() { |
| while(read\_list(in\_order)) { |
| read\_list(post\_order); |
| build(0, n-1, 0, n-1); |
| best\_sum = 1000000000; |
| dfs(post\_order[n-1], 0); |
| cout << best << "\n"; |
| } |
| return 0; |
| } |

1. 二叉树建树

|  |
| --- |
| // UVa122 Trees on the level |
| const int maxn = 256 + 10; |
| struct Node{ |
| bool have\_value; |
| int v; |
| Node\* left, \*right; |
| Node():have\_value(false),left(NULL),right(NULL){} |
| }; |
| Node\* root; |
| Node\* newnode() { return new Node(); } |
| bool failed; |
| void addnode(int v, char\* s) { |
| int n = strlen(s); |
| Node\* u = root; |
| for(int i = 0; i < n; i++) |
| if(s[i] == 'L') { |
| if(u->left == NULL) u->left = newnode(); |
| u = u->left; |
| } else if(s[i] == 'R') { |
| if(u->right == NULL) u->right = newnode(); |
| u = u->right; |
| } |
| if(u->have\_value) failed = true; |
| u->v = v; |
| u->have\_value = true; |
| } |
| void remove\_tree(Node\* u) { |
| if(u == NULL) return; |
| remove\_tree(u->left); |
| remove\_tree(u->right); |
| delete u; } |
| char s[maxn]; |
| bool read\_input() { |
| failed = false; |
| remove\_tree(root); |
| root = newnode(); |
| for(;;) { |
| if(scanf("%s", s) != 1) return false; |
| if(!strcmp(s, "()")) break; |
| int v; |
| sscanf(&s[1], "%d", &v); |
| addnode(v, strchr(s, ',')+1); |
| } |
| return true; } |
| bool bfs(vector<int>& ans) { |
| queue<Node\*> q; |
| ans.clear(); |
| q.push(root); |
| while(!q.empty()) { |
| Node\* u = q.front(); q.pop(); |
| if(!u->have\_value) return false; |
| ans.push\_back(u->v); |
| if(u->left != NULL) q.push(u->left); |
| if(u->right != NULL) q.push(u->right); |
| } |
| return true; } |
| int main() { |
| vector<int> ans; |
| while(read\_input()) { |
| if(!bfs(ans)) failed = 1; |
| if(failed) printf("not complete\n"); |
| else { |
| for(int i = 0; i < ans.size(); i++) { |
| if(i != 0) printf(" "); |
| printf("%d", ans[i]); |
| } |
| printf("\n"); |
| } } return 0; } |

* + 1. saff

1. 其他
   1. C++大数

|  |
| --- |
| struct Bign |
| { |
| int len,s[MAXN]; |
| Bign(){ |
| memset(s,0,sizeof(s)); |
| len=1; |
| } |
| Bign(int num){\*this=num;} |
| Bign(const char \*num){\*this=num;} |
| void clean(){while(len>1&&!s[len-1])len--;} |
| Bign operator = (const int num){ |
| char s[MAXN]; |
| sprintf(s,"%d",num); |
| \*this=s; |
| return \*this; |
| } |
| Bign operator = (const char \*num) { |
| len=strlen(num); |
| for(int i=0;i<len;i++)s[i]=num[len-i-1]-'0'; |
| return \*this; |
| } |
| Bign operator + (const Bign& b) { |
| Bign c; |
| c.len=0; |
| for(int i=0,g=0;g||i<Max(len,b.len);i++){ |
| int x=g; |
| if(i<b.len)x+=b.s[i]; |
| if(i<len)x+=s[i]; |
| c.s[c.len++]=x%10; |
| g=x/10; |
| } |
| return c; |
| } |
| Bign operator - (const Bign& b){ |
| Bign c; |
| c.len=0; |
| for(int i=0,g=0;i<len;i++){ |
| int x=s[i]-g; |
| if(i<b.len)x-=b.s[i]; |
| if(x>=0)g=0; |
| else{g=1;x+=10;} |
| c.s[c.len++]=x; |
| } |
| c.clean(); |
| return c; |
| } |
| Bign operator \* (const Bign& b){ |
| Bign c; |
| c.len=len+b.len; |
| for(int i=0;i<len;i++){ |
| for(int j=0;j<b.len;j++) c.s[i+j]+=s[i]\*b.s[j]; |
| } |
| for(int i=0;i<c.len;i++) { |
| c.s[i+1]+=c.s[i]/10; |
| c.s[i]%=10; |
| } |
| c.clean(); |
| return c; |
| } |
| Bign operator \* (const int& b){ |
| Bign c; |
| c.len=0; |
| for(int i=0,g=0;g||i<len;i++){ |
| int x; |
| if(i<len)x=s[i]\*b+g; |
| else x=g; |
| c.s[c.len++]=x%10; |
| g=x/10; |
| } |
| return c; |
| } |
| Bign operator / (const Bign& b){ |
| Bign c,f=0; |
| for(int i=len-1;i>=0;i--){ |
| f=f\*10; |
| f.s[0]=s[i]; |
| while(f>=b){ |
| f=f-b; |
| c.s[i]++; |
| } |
| } |
| c.len=len; |
| c.clean(); |
| return c; |
| } |
| Bign operator / (const int& b){ |
| Bign c,d=\*this; |
| c.len=len; |
| for(int i=len-1,g=0;i>=0;i--){ |
| d.s[i]+=g\*10; |
| c.s[i]=d.s[i]/b; |
| g=d.s[i]%b; |
| } |
| c.clean(); |
| return c; |
| } |
| Bign operator % (const Bign& b){ |
| Bign c=\*this/b; |
| c=\*this-c\*b; |
| return c; |
| } |
| Bign operator += (const Bign& b) |
| {\*this=\*this+b;return \*this;} |
| Bign operator -= (const Bign& b) |
| {\*this=\*this-b;return \*this;} |
| Bign operator \*= (const Bign& b) |
| {\*this=\*this\*b;return \*this;} |
| Bign operator /= (const Bign& b) |
| {\*this=\*this/b;return \*this;} |
| Bign operator \*= (const int& b) |
| {\*this=\*this\*b;return \*this;} |
| Bign operator /= (const int& b) |
| {\*this=\*this/b;return \*this;} |
| Bign operator %= (const Bign& b) |
| {\*this=\*this%b;return \*this;} |
| bool operator < (const Bign& b){ |
| if(b.len!=len)return len<b.len; |
| for(int i=len-1;i>=0;i--) if(s[i]!=b.s[i])return s[i]<b.s[i]; |
| return 0; |
| } |
| bool operator > (const Bign& b){ |
| if(b.len!=len)return len>b.len; |
| for(int i=len-1;i>=0;i--) if(s[i]!=b.s[i])return s[i]>b.s[i]; |
| return 0; |
| } |
| bool operator == (const Bign& b) |
| {return !(\*this>b)&&!(\*this<b);} |
| bool operator <= (const Bign& b) |
| {return !(\*this>b);} |
| bool operator >= (const Bign& b) |
| {return !(\*this<b);} |
| bool operator != (const Bign& b) |
| {return !(\*this==b);} |
| string str() const { |
| string res; |
| for(int i=0;i<len;i++) |
| res=char(s[i]+'0')+res; |
| return res; |
| } |
| }; |
| //cin 读入 |
| istream& operator >> (istream&in,Bign &x){ |
| string s; |
| in>>s; |
| x=s.c\_str(); |
| return in; |
| } |
| ostream& operator << (ostream&out,Bign x){ |
| out<<x.str(); |
| return out; |
| } |
| int main(){ |
| Bign a,b; |
| cin>>a>>b; |
| cout<<a%b<<endl; |
| return 0; |
| } |

* 1. 基姆拉尔森公式,给年月日,计算星期几

W = (D + 2 \* M + 3 \* (M + 1) \ 5 + Y + Y \ 4 - Y \ 100 + Y \ 400) Mod 7;

* 1. fsfs

1. 图
   1. AStar\_K短路

|  |
| --- |
| const int maxn=100010; |
| int n,m,dis[maxn]; |
| int tot,head1[maxn],head2[maxn]; |
| bool flag[maxn]; |
| struct edge{ |
| int to; |
| int w; |
| int next; |
| }e[maxn\*2],e2[maxn\*2]; |
| struct node{ |
| int f; |
| int g; |
| int from; |
| bool operator < (node a)const{ |
| if(a.f==f) |
| return g>a.g; |
| return f>a.f; |
| } |
| }; |
| void add\_edge(int u,int v,int w){ |
| tot++; |
| e[tot].to=v; |
| e[tot].w=w; |
| e[tot].next=head1[u]; |
| head1[u]=tot; |
| e2[tot].to=u; |
| e2[tot].w=w; |
| e2[tot].next=head2[v]; |
| head2[v]=tot; |
| } |
| void prepare(){ |
| for(int i=1;i<=n;i++) |
| dis[i]=maxn;tot=0; |
| memset(head1,0,sizeof(head1)); |
| memset(head2,0,sizeof(head2)); |
| } |
| void spfa(int t){ |
| for(int i=1;i<=n;i++) |
| dis[i]=maxn; |
| dis[t]=0; |
| queue<int> q; |
| q.push(t); |
| flag[t]=1; |
| while(!q.empty()){ |
| int v=q.front(); |
| q.pop();flag[v]=0; |
| for(int i=head2[v];i;i=e2[i].next) |
| if(dis[e2[i].to]>dis[v]+e2[i].w){ |
| dis[e2[i].to]=dis[v]+e2[i].w; |
| if(!flag[e2[i].to]){ |
| q.push(e2[i].to); |
| flag[e2[i].to]=1; |
| } |
| } |
| } |
| } |
| int a\_star(int s,int t,int k){ |
| if(s==t) k++; |
| if(dis[s]==maxn) return -1; |
| priority\_queue<node> q; |
| int cnt=0; |
| node tmp,to; |
| tmp.from=s; |
| tmp.g=0; |
| tmp.f=tmp.g+dis[tmp.from]; |
| q.push(tmp); |
| while(!q.empty()){ |
| tmp=q.top(); |
| q.pop(); |
| if(tmp.from==t) cnt++; |
| if(cnt==k) return tmp.g; |
| for(int i=head1[tmp.from];i;i=e[i].next){ |
| to.from=e[i].to; |
| to.g=tmp.g+e[i].w; |
| to.f=to.g+dis[to.from]; |
| q.push(to); |
| } |
| } |
| return -1; |
| } |
| int main(){ // 该模板能处理带环图 |
| int x,y,z,s,t,k; |
| while(cin>>n>>m) {// 输入n个点 m条边 |
| prepare(); |
| cin>>s>>t>>k; // 输入起点 终点 第k短路 |
| for(int i=1;i<=m;i++) {// 输入边 |
| cin>>x>>y>>z; |
| add\_edge(x,y,z); |
| } |
| spfa(t); |
| int ans=a\_star(s,t,k); // ans 为第k短路的长度 |
| } |
| return 0; |
| } |

* 1. DAG深度优先队列标记

|  |
| --- |
| /\*DAG(有向无环图)的深度优先搜索标记 |
| \* INIT:edge[][]邻接矩阵；pre[], post[], tag全置0 |
| CALL:dfsTag(i, n); pre/post:开始/结束时间\*/ |
| const int V = 1010; |
| int edge[V][V]; |
| int pre[V]; |
| int post[V]; |
| int tag; |
| void dfsTag(int cur, int n){ |
| //vertex:0 ~ n - 1 |
| pre[cur] = ++tag; |
| for (int i = 0; i < n; i++){ |
| if (edge[cur][i]){ |
| if (0 == pre[i]){ |
| std::cout << "Three Edge!" << '\n'; |
| dfsTag(i, n); |
| } |
| else{ |
| if (0 == post[i]) std::cout << "Back Edge!" << '\n'; |
| else if (pre[i] > pre[cur]) std::cout << "Down Edge!" << '\n'; |
| else std::cout << "Cross Edge!" << '\n'; |
| }} |
| } |
| post[cur] = ++tag; |
| return ; |
| } |

* 1. 无向图找桥

|  |
| --- |
| /\*无向图找桥 |
| \* INIT: edge[][]邻接矩阵；vis[],pre[],ans[],bridge置0； |
| CALL: dfs(0, -1, 1, n);\*/ |
| const int V = 1010; |
| int bridge; //桥 |
| int edge[V][V]; |
| int ans[V]; |
| int pre[V]; |
| int vis[V]; |
| void dfs(int cur, int father, int dep, int n){ |
| //vertex: 0 ~ n - 1 |
| if (bridge) return ; |
| vis[cur] = 1; |
| pre[cur] = ans[cur] = dep; |
| for (int i = 0; i < n; i++){ |
| if (edge[cur][i]){ |
| if (i != father && 1 == vis[i]){ |
| if (pre[i] < ans[cur]) ans[cur] = pre[i]; //back edge |
| } |
| if (0 == vis[i]) { //tree edge |
| dfs(i, cur, dep + 1, n); |
| if (bridge) return ; |
| if (ans[i] < ans[cur]) ans[cur] = ans[i]; |
| if (ans[i] > pre[cur]){bridge = 1; return ;} |
| } |
| } |
| } |
| vis[cur] = 2; |
| } |
| int main(){ |
| // 在这里输入n |
| /\* |
| \* 在这里输入图 |
| \*/ |
| // dfs(0,-1,1,n); 调用函数 |
| } |

* 1. 无向图连通度(割点)

|  |
| --- |
| const int V = 1010; |
| int edge[V][V]; |
| int anc[V]; |
| int pre[V]; |
| int vis[V]; |
| int deg[V]; |
| void dfs(int cur, int father, int dep, int n){ |
| //vertex:0 ~ n - 1 |
| int cnt = 0; |
| vis[cur] = 1; |
| pre[cur] = anc[cur] = dep; |
| for (int i = 0; i < n; i++){ |
| if (edge[cur][i]){ |
| if (i != father && 1 == vis[i]) if (pre[i] < anc[cur]) anc[cur] = pre[i]; //back edge |
| if (0 == vis[i]){ //tree edge |
| dfs(i, cur, dep + 1, n); |
| cnt++; //分支个数 |
| if (anc[i] < anc[cur]) anc[cur] = anc[i]; |
| if ((cur == 0 && cnt > 1) || (cnt != 0 && anc[i] >= pre[cur])) deg[cur]++; //link degree of a vertex |
| } |
| } |
| } |
| vis[cur] = 2; |
| } |
| int main(){ |
| /\* INIT: edge[][]邻接矩阵；vis[],pre[],anc[],deg[]置为0； |
| \* CALL: dfs(0, -1, 1, n); |
| \* k = deg[0], deg[i] + 1(i = 1...n - 1)为删除该节点后得到的连通图个数 |
| 注意: 0作为根比较特殊\*/ |
| } |
| const int V = 1010; |
| int edge[V][V]; |
| int anc[V]; |
| int pre[V]; |
| int vis[V]; |
| int deg[V]; |
| void dfs(int cur, int father, int dep, int n){ |
| //vertex:0 ~ n - 1 |
| int cnt = 0; |
| vis[cur] = 1; |
| pre[cur] = anc[cur] = dep; |
| for (int i = 0; i < n; i++){ |
| if (edge[cur][i]){ |
| if (i != father && 1 == vis[i]) if (pre[i] < anc[cur]) anc[cur] = pre[i]; //back edge |
| if (0 == vis[i]){ //tree edge |
| dfs(i, cur, dep + 1, n); |
| cnt++; //分支个数 |
| if (anc[i] < anc[cur]) anc[cur] = anc[i]; |
| if ((cur == 0 && cnt > 1) || (cnt != 0 && anc[i] >= pre[cur])) deg[cur]++; //link degree of a vertex |
| } |
| } |

* 1. 曼哈顿最小生成树

|  |
| --- |
| const int MAXN = 100010; |
| const int INF = 0x3f3f3f3f; |
| struct Point{ |
| int x; |
| int y; |
| int id; |
| }poi[MAXN]; |
| bool cmp(Point a, Point b){ |
| if (a.x != b.x) return a.x < b.x; |
| else return a.y < b.y; |
| } |
| //树状数组，找y - x大于当前的，但是y + x最小的 |
| struct BIT{ |
| int minVal; |
| int pos; |
| void init(){ |
| minVal = INF; |
| pos = -1; |
| } |
| }bit[MAXN]; |
| //所有有效边 |
| struct Edge{ |
| int u; |
| int v; |
| int d; |
| }edge[MAXN << 2]; |
| bool cmpEdge(Edge a, Edge b){ return a.d < b.d;} |
| int tot; |
| int n; |
| int F[MAXN]; |
| int find(int x){ |
| if (F[x] == -1) return x; |
| else return F[x] = find(F[x]); |
| } |
| void addEdge(int u, int v, int d){ |
| edge[tot].u = u; |
| edge[tot].v = v; |
| edge[tot++].d = d; |
| return ; |
| } |
| int lowbit(int x){ return x & (-x);} |
| //更新bit |
| void update(int i, int val, int pos){ |
| while (i > 0){ |
| if (val < bit[i].minVal){ |
| bit[i].minVal = val; |
| bit[i].pos = pos; |
| } |
| i -= lowbit(i); |
| } |
| return ; |
| } |
| //查询[i, m]的最小值位置 |
| int ask(int i, int m){ |
| int minVal = INF; |
| int pos = -1; |
| while (i <= m){ |
| if (bit[i].minVal < minVal){ |
| minVal = bit[i].minVal; |
| pos = bit[i].pos; |
| } |
| i += lowbit(i); |
| } |
| return pos; |
| } |
| int dist(Point a, Point b){ return abs(a.x - b.x) + abs(a.y - b.y);} |
| void ManhattanMinimumSpanningTree(int n, Point p[]){ |
| int a[MAXN], b[MAXN]; |
| tot = 0; |
| for (int dir = 0; dir < 4; dir++){ |
| //变换4种坐标 |
| if (dir == 1 || dir == 3){ |
| for (int i = 0; i < n; i++) std::swap(p[i].x, p[i].y); |
| } |
| else if (dir == 2){ |
| for (int i = 0; i < n; i++) p[i].x = -p[i].x; |
| } |
| std::sort(p, p + n, cmp); |
| for (int i = 0; i < n; i++) a[i] = b[i] = p[i].y - p[i].x; |
| std::sort(b, b + n); |
| int m = (int)(std::unique(b, b + n) - b); |
| for (int i = 1; i <= m; i++) bit[i].init(); |
| for (int i = n - 1; i >= 0; i--){ |
| int pos = (int)(std::lower\_bound(b, b + m, a[i]) - b + 1); |
| int ans = ask(pos, m); |
| if (ans != -1) addEdge(p[i].id, p[ans].id, dist(p[i], p[ans])); |
| update(pos, p[i].x + p[i].y, i); |
| } |
| } |
| return ; |
| } |
| int solve(int k){ |
| ManhattanMinimumSpanningTree(n, poi); |
| memset(F, -1, sizeof(F)); |
| std::sort(edge, edge + tot, cmpEdge); |
| for (int i = 0; i < tot; i++){ |
| int u = edge[i].u; |
| int v = edge[i].v; |
| int tOne = find(u); |
| int tTwo = find(v); |
| if (tOne != tTwo){ |
| F[tOne] = tTwo; |
| k--; |
| if (k == 0) return edge[i].d; |
| } |
| } |
| return -1; |
| } |
| int main(int argc, const char \* argv[]){ |
| //freopen("in.txt", "r", stdin); |
| //freopen("out.txt", "w", stdout); |
| int k; |
| while ((std::cin >> n >> k) && n){ |
| for (int i = 0; i < n; i++){ |
| std::cin >> poi[i].x >> poi[i].y; |
| poi[i].id = i; |
| } |
| std::cout << solve(n - k) << std::endl; |
| } |
| return 0; |
| } |

* 1. 最小生成树(prim)

|  |
| --- |
| #define inf 0x3f3f3f3f |
| typedef struct { |
| int point; |
| int value; |
| }node; |
| int N; |
| int sum; |
| vector<node>point[1000]; |
| int hashtable[1000] = {0}; |
| int num[1000]; |
| void init() { |
| int i; |
| for (i = 0; i < N; i++) num[i] = inf; |
| num[0] = 0; |
| sum = 0; |
| } |
| void prim() { |
| int n = N,i,k,min,min\_num; |
| while (n--) { |
| min=-1; |
| while (n--) { |
| min\_num = inf; |
| for (i = 0; i < N; i++) { |
| if (hashtable[i] == 0 && num[i] != inf) { |
| if (num[i] < min\_num) { |
| Min=i; |
| min\_num = num[i]; |
| } |
| } |
| } |
| sum += num[min]; |
| hashtable[min] = 1; |
| if (min == -1) return; |
| for (k = 0; k < point[min].size(); k++) { |
| int v = point[min][k].point; |
| int value = point[min][k].value; |
| if (num[v] > value && hashtable[v] == 0) num[v] = value; |
| } |
| } |
| } |
| int main() { |
| int M; |
| scanf("%d", &N); |
| scanf("%d", &M); |
| int x, y, i, check, value; |
| for (i = 0; i < M; i++) { |
| scanf("%d%d%d", &x, &y, &value); |
| node new\_node = { y,value }; |
| point[x].push\_back(new\_node); |
| new\_node.point = x; |
| point[y].push\_back(new\_node); |
| } |
| init(); |
| prim(); |
| printf("%d",sum); |
| scanf("%d", &check); |
| } |

* 1. 次小生成树

|  |
| --- |
| int g[M][M],path[M][M];//path求的是i到j最大的边权 |
| int dist[M],pre[M],vis[M]; |
| bool used[M][M];//是否在最小生成树中 |
| int n,m,mst; |
| void init(){ |
| for(int i=0;i<=n;i++) |
| for(int j=i+1;j<=n;j++) g[i][j]=g[j][i]=inf; |
| } |
| int prime() { |
| int mst=0; |
| memset(path,0,sizeof(path)); |
| memset(vis,0,sizeof(vis)); |
| memset(used,0,sizeof(used)); |
| vis[1]=1; |
| for(int i=1;i<=n;i++){ |
| dist[i]=g[1][i]; |
| pre[i]=1; |
| } |
| for(int i=1;i<n;i++) { |
| int u=-1; |
| for(int j=1;j<=n;j++){ |
| if(!vis[j]) if(u==-1||dist[j]<dist[u]) u=j; |
| } |
| used[u][pre[u]]=used[pre[u]][u]=true;//加入mst |
| mst+=g[pre[u]][u]; |
| vis[u]=1; |
| for(int j=1;j<=n;j++) |
| { |
| if(vis[j]&&j!=u)//从u到j这条路径上最大边的权值 |
| path[j][u]=path[u][j]=max(path[j][pre[u]],dist[u]); |
| if(!vis[j]) |
| if(dist[j]>g[u][j]){//更新相邻节点的距离 |
| dist[j]=g[u][j]; |
| pre[j]=u;//记录他的前驱 |
| } |
| } |
| } |
| return mst; |
| } |
| int second\_tree(){//求次小生成树 |
| int res=inf; |
| for(int i=1;i<=n;i++) |
| for(int j=1;j<=n;j++) |
| if(i!=j&&!used[i][j]) |
| res=min(res,mst-path[i][j]+g[i][j]);//删除树上权值最大的路径并且加上这条路径其它边 |
| return res; |
| } |
| int main() { |
| int t; |
| scanf("%d",&t); |
| while(t--) { |
| scanf("%d%d",&n,&m); |
| init(); |
| mst=prime();//最小生成树 |
| int second\_mst=second\_tree();//次小生成树 |
| } |
| } |

* 1. 欧拉路径

|  |
| --- |
| /\*SGU 101 \*/ |
| struct Edge{ |
| int to; |
| int next; |
| int index; |
| int dir; |
| bool flag; |
| } edge[220]; |
| int head[10]; //前驱 |
| int tot; |
| void init(){ |
| memset(head, -1, sizeof((head))); |
| tot = 0; |
| } |
| void addEdge(int u, int v, int index){ |
| edge[tot].to = v; |
| edge[tot].next = head[u]; |
| edge[tot].index = index; |
| edge[tot].dir = 0; |
| edge[tot].flag = false; |
| head[u] = tot++; |
| edge[tot].to = u; |
| edge[tot].next = head[v]; |
| edge[tot].index = index; |
| edge[tot].dir = 1; |
| edge[tot].flag = false; |
| head[v] = tot++; |
| return ; |
| } |
| int du[10]; |
| std::vector<int>ans; |
| void dfs(int u){ |
| for (int i = head[u]; i != -1; i = edge[i].next){ |
| if (!edge[i].flag){ |
| edge[i].flag = true; |
| edge[i ^ 1].flag = true; |
| dfs(edge[i].to); |
| ans.push\_back(i); //容器尾部插入i |
| } |
| } |
| return ; |
| } |
| int main(){ |
| //freopen("in.txt", "r", stdin); |
| //freopen("out.txt", "w", stdout); |
| int n; |
| while (std::cin >> n) { |
| init(); |
| int u, v; |
| memset(du, 0, sizeof(du)); |
| for (int i = 1; i <= n; i++){ |
| std::cin >> u >> v; |
| addEdge(u, v, i); |
| du[u]++; |
| du[v]++; |
| } |
| int s = -1; |
| int cnt = 0; |
| for (int i = 0; i <= 6; i++){ |
| if (du[i] & 1){ |
| cnt++; |
| s = i; |
| } |
| if (du[i] > 0 && s == -1) s = i; |
| } |
| if (cnt != 0 && cnt != 2){ |
| std::cout << "No solution" << '\n'; |
| continue; |
| } |
| ans.clear(); |
| dfs(s); |
| if (ans.size() != n){ |
| std::cout << "No solution" << '\n'; |
| continue; |
| } |
| for (int i = 0; i < ans.size(); i++){ |
| printf("%d ", edge[ans[i]].index); |
| if (edge[ans[i]].dir == 0) std::cout << "-" << '\n'; |
| else std::cout << "+" << '\n'; |
| } |
| } |
| return 0; |
| } |

* 1. 迪杰斯特拉优化模板

|  |
| --- |
| typedef struct { |
| int point;//能够到达的点 |
| int value;//第一尺度 |
| int cost; //第二尺度 |
| }node; |
| int N; |
| int num[1001];//第一尺度的最小值储存单位 |
| int cost[1001];//第二尺度的最小值储存单位 |
| int hashtable[2000] = { 0 };//哈希表，判断点是否访问过 |
| vector<node>point[2000];//邻接表 |
| void init(int start) {//初始化 |
| int i; |
| for (i = 0; i < N; i++) { |
| num[i] = Max; |
| cost[i] = Max; |
| } |
| num[start] = 0; |
| cost[start] = 0; |
| } |
| void djistra(int start) { |
| init(start); |
| int i; |
| int min = 0; |
| int min\_num; |
| int check; |
| while (1) { |
| min\_num = Max; |
| check = 0; |
| for (i = 0; i < N; i++) {//找出当前离起点最近的且未访问过的节点 |
| if (hashtable[i] == 0 && num[i] != Max) { |
| check = 1; |
| if (num[i] < min\_num) { |
| min = i; |
| min\_num = num[i]; |
| } |
| } |
| } |
| if (check == 0) |
| return;//如果没有就说明优化距离结束 |
| hashtable[min] = 1; |
| for (i = 0; i < point[min].size(); i++) { |
| if (hashtable[point[min][i].point] == 0) { |
| if (num[point[min][i].point] > point[min][i].value + num[min]) {//以第一尺度为标准，先计算出第一尺度的最小值下的第二尺度的值 |
| num[point[min][i].point] = point[min][i].value + num[min]; |
| cost[point[min][i].point] = point[min][i].cost + cost[min]; |
| } |
| else if (num[point[min][i].point] == point[min][i].value + num[min]) {//以计算出的第二尺度值为标准，计算出第二尺度的最小值 |
| if (cost[point[min][i].point] > point[min][i].cost + cost[min]) |
| cost[point[min][i].point] = point[min][i].cost + cost[min]; |
| } |
| } |
| } |
| } |
| } |
| int main() { //点标号0开头 |
| int M, start, end; |
| int x, y, value,cost\_value; |
| while (scanf("%d%d", &N, &M) && (N != 0 || M != 0) ) { |
| while (M--) { |
| scanf("%d%d%d%d", &x, &y, &value, &cost\_value); |
| node new\_node = { y,value,cost\_value }; |
| point[x].push\_back(new\_node);//无向边 |
| new\_node.point = x; |
| point[y].push\_back(new\_node); |
| } |
| scanf("%d%d", &start, &end); |
| djistra(start); |
| printf("%d %d\n", num[end], cost[end]); |
| } |
| } |

* 1. Dfja

1. 字符串
   1. a字符串增删改变b字符串的最小操作数

|  |
| --- |
| const int N = 1e3 + 5; |
| int T, cas = 0; |
| int n, m; |
| int dp[N][N]; |
| char s[N], t[N]; |
| int main(){ |
| while (scanf("%s%s", s, t) != EOF){ |
| int n = (int)strlen(s), m = (int)strlen(t); |
| for (int i = 0; i <= n; i++) dp[i][0] = i; |
| for (int i = 0; i <= m; i++) dp[0][i] = i; |
| for (int i = 1; i <= n; i++){ |
| for (int j = 1; j <= m; j++){ |
| dp[i][j] = min(dp[i - 1][j], dp[i][j - 1]) + 1; |
| dp[i][j] = min(dp[i][j], dp[i - 1][j - 1] + (s[i - 1] != t[j - 1])); |
| } |
| } |
| printf("%d\n", dp[n][m]); |
| } |
| return 0; |
| } |

* 1. KMP

|  |
| --- |
| /\*字符串匹配。给你两个字符串，寻找其中一个字符串是否包含另一个字符串， |
| 如果包含，返回包含的起始位置。\*/ |
| /\* next[]的含义，x[i - next[i]...i - 1] = x[0...next[i] - 1] |
| \* next[i]为满足x[i - z...i - 1] = x[0...z - 1]的最大z值（就是x的自身匹配）\*/ |
| void KMP\_Pre(char x[], int m, int next[]){ |
| int i, j; |
| j = next[0] = -1; |
| i = 0; |
| while (i < m){ |
| while (-1 != j && x[i] != x[j]) j = next[j]; |
| next[++i] = ++j; |
| } |
| return ; |
| } |
| /\* kmpNext[]的意思：next'[i] = next[next[...[next[i]]]] |
| \* （直到next'[i] < 0或者x[next'[i]] != x[i]） |
| \* 这样的预处理可以快一些 \*/ |
| void preKMP(char x[], int m, int kmpNext[]){ |
| int i, j; |
| j = kmpNext[0] = -1; |
| i = 0; |
| while (i < m){ |
| while (-1 != j && x[i] != x[j]) j = kmpNext[j]; |
| if (x[++i] == x[++j]) kmpNext[i] = kmpNext[j]; |
| else kmpNext[i] = j; |
| } |
| return ; |
| } |
| /\*此函数与上述两个函数中的任意一个搭配使用（即调用上述两个函数中的任意一个） |
| \* 返回x在y中出现的次数，可以重叠 \*/ |
| int next[10010]; |
| int KMP\_Count(char x[], int m, char y[], int n){ |
| // x是模式串，y是主串 |
| int i, j; |
| int ans = 0; |
| // preKMP(x, m, next); |
| KMP\_Pre(x, m, next); |
| i = j = 0; |
| while (i < n){ |
| while (-1 != j && y[i] != x[j]) j = next[j]; |
| i++, j++; |
| if (j >= m){ |
| ans++; |
| j = next[j]; |
| } |
| } |
| return ans; |
| } |
| int main(){ |
| int ans = 0; |
| char y[] = "ccabcabcaa"; |
| char a[] = "abc"; |
| int leny = strlen(y); |
| int lena = strlen(a); |
| ans = KMP\_Count(a,lena,y,leny);//统计a在y中出现的次数 |
| cout << ans << endl; |
| } |

* 1. 扩展kmp

|  |
| --- |
| int IT\_MAX = 1 << 19; |
| int MOD = 1000000007; |
| const int INF = 0x3f3f3f3f; |
| const ll LL\_INF = 0x3f3f3f3f3f3f3f3f; |
| const db PI = acos(-1); |
| const db ERR = 1e-10; |
| const int MAX\_N = 100005; |
| bool cmp (int a , int b){ |
| /\*扩展KMP解决的问题： |
| 定义母串S和子串T，S的长度为n，T的长度为m； |
| 求字符串T与字符串S的每一个后缀的最长公共前缀； |
| 也就是说，设有extend数组： |
| extend[i]表示T与S[i,n-1]的最长公共前缀，要求出所有extend[i](0<=i<n)。\*/ |
| /\*扩展KMP |
| \* next[i]:x[i...m-1]的最长公共前缀 |
| \* extend[i]:y[i...n-1]与x[0...m-1]的最长公共前缀 \*/ |
| void preEKMP(char x[], int m, int next[]){ |
| next[0] = m; |
| int j = 0; |
| while (j + 1 < m && x[j] == x[j + 1]) j++; |
| next[1] = j; |
| int k = 1; |
| for (int i = 2; i < m; i++){ |
| int p = next[k] + k - 1; |
| int L = next[i - k]; |
| if (i + L < p + 1) next[i] = L; |
| else{ |
| j = std::max(0, p - i + 1); |
| while (i + j < m && x[i + j] == x[j]) j++; |
| next[i] = j; |
| k = i; |
| } |
| } |
| return ; |
| } |
| void EKMP(char x[], int m, char y[], int n, int next[], int extend[]){ |
| preEKMP(x, m, next); |
| int j = 0; |
| while (j < n && j < m && x[j] == y[j]) j++; |
| extend[0] = j; |
| int k = 0; |
| for (int i = 1; i < n; i++){ |
| int p = extend[k] + k - 1; |
| int L = next[i - k]; |
| if (i + L < p + 1) extend[i] = L; |
| else{ |
| j = std::max(0, p - i + 1); |
| while (i + j < n && j < m && y[i + j] == x[j]) j++; |
| extend[i] = j; |
| k = i; |
| } |
| } |
| return ; |
| } |
| int main(){ |
| char y[] = "aaaabcaaaaa"; |
| char a[] = "aaaaa"; |
| int next[1000],extend[1000]; |
| int leny = strlen(y); |
| int lena = strlen(a); |
| EKMP(a,lena,y,leny,next,extend) ; |
| for(int i = 0 ; i < 10 ; i ++) cout << next[i] << " "; |
| } |

* 1. 最短公共祖先

|  |
| --- |
| /\*求解类似于 包含 "alba"和"baccdasd" 最短多长的问题; |
| 答案为 len("albaccdasd") == 10; \*/ |
| const int N = 1000010; |
| char a[2][N]; |
| int fail[N]; |
| inline int max(int a, int b){ return (a > b) ? a : b;} |
| int kmp(int &i, int &j, char\* str, char\* pat){ |
| int k; |
| memset(fail, -1, sizeof(fail)); |
| for (i = 1; pat[i]; ++i){ |
| for (k = fail[i - 1]; k >= 0 && pat[i] != pat[k + 1]; k = fail[k]); |
| if (pat[k + 1] == pat[i]) fail[i] = k + 1; |
| } |
| i = j = 0; |
| while (str[i] && pat[j]){ |
| if (pat[j] == str[i]){ i++; |
| else if (j == 0) i++; // 第一个字符匹配失败，从str下一个字符开始 |
| else j = fail[j - 1] + 1; |
| } |
| if (pat[j]) return -1; |
| else return i - j; |
| } |
| int main(int argc, const char \* argv[]){ |
| int T; |
| scanf("%d", &T); |
| while (T--){ |
| int i, j, l1 = 0, l2 = 0; |
| cin >> a[0] >> a[1]; |
| int len1 = (int)strlen(a[0]), len2 = (int)strlen(a[1]), val; |
| val = kmp(i, j, a[1], a[0]); // a[1]在前 |
| if (val != -1) l1 = len1; |
| else{ |
| //printf("i:%d, j:%d\n", i, j); |
| if (i == len2 && j - 1 >= 0 && a[1][len2 - 1] == a[0][j - 1]) l1 = j; |
| } |
| val = kmp(i, j, a[0], a[1]); // a[0]在前 |
| if (val != -1) l2 = len2; |
| else{ |
| //printf("i:%d, j:%d\n", i, j); |
| if (i == len1 && j - 1 >= 0 && a[0][len1 - 1] == a[1][j - 1]) l2 = j; |
| } |
| //printf("l1:%d,l2:%d\n",l1,l2); |
| printf("%d\n", len1 + len2 - max(l1, l2)); |
| } |
| return 0; |
| } |

* 1. AC

|  |
| --- |
| /\*给你一个字典树,再给你一个字符串 , 查询树上的东西出现了几次 \*/ |
| struct Trie{ |
| int next[500010][26], fail[500010], end[500010]; |
| int root, L; |
| int newnode(){ |
| for (int i = 0; i < 26; i++) next[L][i] = -1; |
| end[L++] = 0; |
| return L - 1; |
| } |
| void init(){ |
| L = 0; |
| root = newnode(); |
| } |
| void insert(char buf[]){ |
| int len = (int)strlen(buf); |
| int now = root; |
| for (int i = 0; i < len; i++){ |
| if (next[now][buf[i] - 'a'] == -1) next[now][buf[i] - 'a'] = newnode(); |
| now = next[now][buf[i] - 'a']; |
| } |
| end[now]++; |
| } |
| void build(){ |
| queue<int>Q; |
| fail[root] = root; |
| for (int i = 0; i < 26; i++){ |
| if (next[root][i] == -1) next[root][i] = root; |
| else{ |
| fail[next[root][i]] = root; |
| Q.push(next[root][i]); |
| } |
| } |
| while (!Q.empty()){ |
| int now = Q.front(); |
| Q.pop(); |
| for (int i = 0;i < 26;i++){ |
| if (next[now][i] == -1) next[now][i] = next[fail[now]][i]; |
| else{ |
| fail[next[now][i]]=next[fail[now]][i]; |
| Q.push(next[now][i]); |
| } |
| } |
| } |
| } |
| int query(char buf[]){ |
| int len = (int)strlen(buf); |
| int now = root; |
| int res = 0; |
| for (int i = 0; i < len; i++){ |
| now = next[now][buf[i] - 'a']; |
| int temp = now; |
| while (temp != root){ |
| res += end[temp]; |
| end[temp] = 0; |
| temp = fail[temp]; |
| } |
| } |
| return res; |
| } |
| void debug(){ |
| for (int i = 0; i < L; i++){ |
| printf("id = %3d,fail = %3d,end = %3d,chi = [", i, fail[i], end[i]); |
| for (int j = 0; j < 26; j++) printf("%2d", next[i][j]); |
| printf("]\n"); |
| } |
| } |
| }; |
| char buf[1000010]; |
| Trie ac; |
| int main(){ |
| int T; |
| int n; |
| scanf("%d", &T); |
| while(T--){ |
| scanf("%d", &n); |
| ac.init(); |
| for (int i = 0; i < n; i++){ |
| scanf("%s", buf); |
| ac.insert(buf); |
| } |
| ac.build(); |
| scanf("%s", buf); |
| printf("%d\n", ac.query(buf)); |
| } |
| return 0; |
| } |

* 1. kmp简易版c++函数

|  |
| --- |
| /\* strstr函数 |
| \* 功能：在串中查找指定字符串的第一次出现 |
| \* 用法：char \*strstr(char \*strOne, char \*strTwo); |
| \* 据说strstr函数和KMP的算法效率差不多\*/ |
| int main(int argc, const char \* argv[]){ |
| char strOne[] = "Borland International"; |
| char strTwo[] = "nation"; |
| char \*ptr; |
| ptr = strstr(strOne, strTwo); |
| std::cout << ptr << '\n'; |
| return 0; |
| } |

* 1. KR字符串匹配(预处理,kmp失败可用)

|  |
| --- |
| // Rabin Karp Algorithm |
| /\*未调试 \*/ |
| void Rabin\_Karp\_search(const string &T, const string &P, int d, int q){ |
| int m = P.length(); |
| int n = T.length(); |
| int i, j; |
| int p = 0; // hash value for pattern |
| int t = 0; // hash value for txt |
| int h = 1; |
| // The value of h would be "pow(d, M-1)%q" |
| for (i = 0; i < m-1; i++) |
| h = (h\*d)%q; |
| // Calculate the hash value of pattern and first window of text |
| for (i = 0; i < m; i++){ |
| p = (d\*p + P[i])%q; |
| t = (d\*t + T[i])%q; |
| } |
| // Slide the pattern over text one by one |
| for (i = 0; i <= n - m; i++){ |
| // Chaeck the hash values of current window of text and pattern |
| // If the hash values match then only check for characters on by one |
| if ( p == t ){ |
| /\* Check for characters one by one \*/ |
| for (j = 0; j < m; j++) if (T[i+j] != P[j]) break; |
| if (j == m) // if p == t and pat[0...M-1] = txt[i, i+1, ...i+M-1] |
| cout<<"Pattern found at index :"<< i<<endl; |
| } |
| // Calulate hash value for next window of text: Remove leading digit, |
| // add trailing digit |
| if ( i < n-m ){ |
| t = (d\*(t - T[i]\*h) + T[i+m])%q; |
| // We might get negative value of t, converting it to positive |
| if(t < 0) t = (t + q); |
| } |
| } |
| } |
| int main(){ |
| string T = "Rabin–Karp string search algorithm: Rabin-Karp"; |
| string P = "Rabin"; |
| int q = 101; // A prime number |
| int d=16; |
| Rabin\_Karp\_search(T,P,d,q); |
| system("pause"); |
| return 0; |
| } |

* 1. 最长回文子串

|  |
| --- |
| /\*求最长回文子串\*/ |
| const int MAXN = 110010; |
| char A[MAXN \* 2]; |
| int B[MAXN \* 2]; |
| void Manacher(char s[], int len){ |
| int l = 0; |
| A[l++] = '$'; //0下标存储为其他字符 |
| A[l++] = '#'; |
| for (int i = 0; i < len; i++){ |
| A[l++] = s[i]; |
| A[l++] = '#'; |
| } |
| A[l] = 0; //空字符 |
| int mx = 0; |
| int id = 0; |
| for (int i = 0; i < l; i++){ |
| B[i] = mx > i ? std::min(B[2 \* id - i], mx - i) : 1; |
| while (A[i + B[i]] == A[i - B[i]]) B[i]++; |
| if (i + B[i] > mx){ |
| mx = i + B[i]; |
| id = i; |
| } |
| } |
| return ; |
| } |
| /\* abaaba |
| \* i: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 |
| \* A: $ # a # b # a # a # b # a # '\0' |
| \* B: 1 1 2 1 4 1 2 7 2 1 4 1 2 1 //以第i个为中心的回文半径（包括第i个）\*/ |
| char s[MAXN]; |
| int main(int argc, const char \* argv[]){ |
| while (std::cin >> s){ |
| int len = (int)strlen(s); |
| Manacher(s, len); |
| int ans = 0; |
| for (int i = 0; i < 2 \* len + 2; i++) { //两倍长度并且首位插有字符，所以i < 2 \* len + 2 |
| ans = std::max(ans, B[i] - 1); |
| } |
| std::cout << ans << std::endl; |
| } |
| return 0; |
| } |

* 1. shhs

1. Fsjla