

- 1 Base algorithm
  - 1.1 Bisection method
- 2 Graph Theory and Network Algorithms
  - 2.1 maxflow
    - 2.1.1 Dinic
    - 2.1.2 ISAP
- 3 Algebraic Algorithms
- 4 Number Theory
- 5 Data structure
- 6 Computational geometry
- 7 Classic Problems

# 1 Base algorithm

---

## 1.1 Bisection method

search for  $\min(b), b \in \{a[k] \geq x\}$

```
1 while(l<r){
2     int mid = (l + r) >> 1;
3     if(a[mid] >= x) r = mid;
4     else l = mid + 1;
5 }
6 return a[l];
```

search for  $\max(b), b \in \{a[k] \leq x\}$

```
1 while(l<r){
2     int mid = (l + r + 1) >> 1;
3     if(a[mid] <= x) l = mid;
4     else r = mid - 1;
5 }
6 return a[l];
```

# 2 Graph Theory and Network Algorithms

---

## 2.1 maxflow

### 2.1.1 Dinic

```
1 class dinic {
2     private:
3         static const int N = 10010; //endpoint_num
4         static const int M = 200010; //edge_num
5         static const int INF = 0x3f3f3f3f;
6         int tot, n, m, s, t;
7         int carc[N]; //curarc
8         int Head[N], nxt[M], ver[M], flow[M]; //base
```

```

9      int d[N]; //depth
10     public:
11         void init(int _n, int _m, int _s, int _t) {
12             tot=1;
13             n=_n, m=_m, s=_s, t=_t;
14             fill(Head, Head+n+1, 0);
15         }
16         void addedge(int u, int v, int w) {
17             ver[++tot]=v;
18             flow[tot]=w;
19             nxt[tot]=Head[u];
20             Head[u]=tot;
21
22             ver[++tot]=u;
23             flow[tot]=0;
24             nxt[tot]=Head[v];
25             Head[v]=tot;
26         }
27         bool bfs() {
28             fill(d, d+n+1, 0);
29             queue<int> q;
30             d[s]=1;
31             q.push(s);
32             while(q.size()) {
33                 int u = q.front();
34                 q.pop();
35                 for(int i = Head[u]; i; i=nxt[i]) {
36                     int v = ver[i];
37                     if(d[v]==0 && flow[i]) {
38                         d[v]=d[u]+1;
39                         q.push(v);
40                     }
41                 }
42             }
43             return d[t]!=0;
44         }
45         int dfs(int u, int minn);
46         int maxflow() {
47             int ans=0;
48             while(bfs()) {
49                 copy(Head+1, Head+n+1, carc+1);
50                 ans+=dfs(s, INF);
51             }
52             return ans;
53         }
54     } flow;
55     int dinic::dfs(int u, int minn) {
56         if(u==t) return minn;
57         int ret=0;
58         for(int i = carc[u]; minn && i; i=nxt[i]) {
59             carc[u]=i;
60             int v = ver[i];
61             if(flow[i] && d[v]==d[u]+1) {
62                 int final=dfs(v, min(flow[i], minn));
63                 if(final>0) {
64                     flow[i]-=final;
65                     flow[i^1]+=final;
66                     minn-=final;

```

```

67         ret+=final;
68     } else d[v]=-1;
69 }
70 }
71 return ret;
72 }

```

## 2.1.2 ISAP

```

1  class ISAP {
2      static const int N = 10010;//endpoint_num
3      static const int M = 240010;//edge_num
4      static const int INF = 0x3f3f3f3f;
5      int tot,n,m,s,t;
6      int carc[N],gap[N];//curarc and gap
7      int pre[N];
8      int Head[N],nxt[M],ver[M],flow[M];//base
9      int d[N];//depth
10     int visit[N];
11     bool visited[N];
12 public:
13     void init(int _n,int _m,int _s,int _t) {
14         tot=1;
15         n=_n,m=_m,s=_s,t=_t;
16         fill(Head,Head+n+1,0);
17         fill(visit,visit+n+1,0);
18     }
19     void addedge(int u,int v,int w) {
20         ver[++tot]=v;
21         flow[tot]=w;
22         nxt[tot]=Head[u];
23         Head[u]=tot;
24
25         ver[++tot]=u;
26         flow[tot]=0;
27         nxt[tot]=Head[v];
28         Head[v]=tot;
29     }
30     bool bfs() { // calculate the depth
31         fill(visited,visited+n+1,0);
32         queue<int>q;
33         visited[t]=1;class ISAP {
34             static const int N = 10010;//endpoint_num
35             static const int M = 240010;//edge_num
36             static const int INF = 0x3f3f3f3f;
37             int tot,n,m,s,t;
38             int carc[N],gap[N];//curarc and gap
39             int pre[N];
40             int Head[N],nxt[M],ver[M],flow[M];//base
41             int d[N];//depth
42             int visit[N];
43             bool visited[N];
44         public:
45             void init(int _n,int _m,int _s,int _t) {
46                 tot=1;

```

```

47     n=_n,m=_m,s=_s,t=_t;
48     fill(Head,Head+n+1,0);
49     fill(visit,visit+n+1,0);
50 }
51 void addedge(int u,int v,int w) {
52     ver[++tot]=v;
53     flow[tot]=w;
54     nxt[tot]=Head[u];
55     Head[u]=tot;
56
57     ver[++tot]=u;
58     flow[tot]=0;
59     nxt[tot]=Head[v];
60     Head[v]=tot;
61 }
62 bool bfs() { // calculate the depth
63     fill(visited,visited+n+1,0);
64     queue<int>q;
65     visited[t]=1;
66     d[t]=0;
67     q.push(t);
68     while(q.size()) {-
69         int u = q.front();
70         q.pop();
71         for(int i = Head[u]; i; i=nxt[i]) {
72             int v = ver[i];
73             if(i&1&&!visited[v]) {
74                 visited[v]=true;
75                 d[v]=d[u]+1;
76                 q.push(v);
77             }
78         }
79     }
80     return visited[s];
81 }
82 int aug() {
83     int u=t,df=INF;
84     while(u!=s) { // calculate the flow
85         df=min(df,flow[pre[u]]);
86         u=ver[pre[u]^1];
87     }
88     u=t;
89
90     while(u!=s) {
91         flow[pre[u]]-=df;
92         flow[pre[u]^1]+=df;
93         u=ver[pre[u]^1];
94     }
95     return df;
96 }
97 int maxflow();
98 } flow;
99 int ISAP :: maxflow() {
100     int ans=0;
101     fill(gap,gap+n+1,0);
102     for(int i=1; i<=n; i++) carc[i]=Head[i]; //copy the head for ignore the
103     useless edge
104     bfs();

```

```

104     for(int i=1; i<=n; i++)gap[d[i]]++;//Using array gap to store how many
        endpoint's depth is k. When we found some gap is 0 or d[source]>n mean
        there are no another augmenting path.
105     int u = s;
106     while(d[s]<=n) {
107         if(u==t) {
108             ans+=aug();
109             u=s;
110         }
111         bool advanced=false;
112         for(int i=carc[u]; i; i=nxt[i]) {
113             if(flow[i]&&d[u]==d[ver[i]]+1) {
114                 advanced=true;
115                 pre[ver[i]]=i;
116                 carc[u]=i;//carc
117                 u=ver[i];
118                 break;
119             }
120         }
121         if(!advanced) {
122             int mindep=n-1;
123             for(int i=Head[u]; i; i=nxt[i]) {
124                 if(flow[i]) {
125                     mindep=min(mindep,d[ver[i]]);
126                 }
127             }
128             if(--gap[d[u]]==0)break;
129             gap[d[u]=mindep+1]++;
130
131             carc[u]=Head[u];
132             if(u!=s)u=ver[pre[u]^1];
133         }
134     }
135     return ans;
136 }
137     d[t]=0;
138     q.push(t);
139     while(q.size()) {
140         int u = q.front();
141         q.pop();
142         for(int i = Head[u]; i; i=nxt[i]) {
143             int v = ver[i];
144             if(i&1&&!visited[v]) {
145                 visited[v]=true;
146                 d[v]=d[u]+1;
147                 q.push(v);
148             }
149         }
150     }
151     return visited[s];
152 }
153 int aug() {
154     int u=t,df=INF;
155     while(u!=s) { // calculate the flow
156         df=min(df, flow[pre[u]]);
157         u=ver[pre[u]^1];
158     }
159     u=t;

```

```

160         while(u!=s) {
161             flow[pre[u]]-=df;
162             flow[pre[u]^1]+=df;
163             u=ver[pre[u]^1];
164         }
165         return df;
166     }
167     int maxflow();
168 } flow;
169 int ISAP :: maxflow() {
170     int ans=0;
171     fill(gap,gap+n+1,0);
172     for(int i=1; i<=n; i++) carc[i]=Head[i]; //copy the head for ignore the
173     useless edge
174     bfs();
175     for(int i=1; i<=n; i++) gap[d[i]]++; //Using array gap to store how many
176     endpoint's depth is k. When we found some gap is 0 or d[source]>n mean
177     there are no another augmenting path.
178     int u = s;
179     while(d[s]<=n) {
180         if(u==t) {
181             ans+=aug();
182             u=s;
183         }
184         bool advanced=false;
185         for(int i=carc[u]; i; i=nxt[i]) {
186             if(flow[i]&& d[u]==d[ver[i]]+1) {
187                 advanced=true;
188                 pre[ver[i]]=i;
189                 carc[u]=i; //carc
190                 u=ver[i];
191                 break;
192             }
193         }
194         if(!advanced) {
195             int mindep=n-1;
196             for(int i=Head[u]; i; i=nxt[i]) {
197                 if(flow[i]) {
198                     mindep=min(mindep,d[ver[i]]);
199                 }
200             }
201             if(--gap[d[u]]==0) break;
202             gap[d[u]=mindep+1]++;
203             carc[u]=Head[u];
204             if(u!=s) u=ver[pre[u]^1];
205         }
206     }
207     return ans;
208 }

```

### 3 Algebraic Algorithms

## **4 Number Theory**

---

## **5 Data structure**

---

## **6 Computational geometry**

---

## **7 Classic Problems**

---