1 Base algorithm

1.1 Bisection method

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

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

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

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

2 Graph Theory and Network Algorithms

2.1 maxflow

2.1.1 Dinic

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

```
    int d[N];//depth
  public:
    void init(int _n,int _m,int _s,int _t) {
        tot=1;
        n=_n,m=_m,s=_s,t=_t;
        fill(Head,Head+n+1,0);
    }
    void addedge(int u,int v,int w) {
        ver[++tot]=v;
        flow[tot]=w;
        nxt[tot]=Head[u];
        Head[u]=tot;

        ver[++tot]=u;
        flow[tot]=0;
        nxt[tot]=Head[v];
        Head[v]=tot;
    }
    bool bfs() {
        fill(d,d+n+1,0);
        queue<int>q;
        d[s]=1;
        q.push(s);
        while(q.size()) {
            int u = q.front();
            q.pop();
            for(int i = Head[u]; i; i=nxt[i]) {
                int v = ver[i];
                if(d[v]==0&&flow[i]) {
                    d[v]=d[u]+1;
                    q.push(v);
                }
            }
        }
        return d[t]!=0;
    }
    int dfs(int u,int minn);
    int maxflow() {
        int ans=0;
        while(bfs()) {
            copy(Head+1,Head+n+1,carc+1);
            ans+=dfs(s,INF);
        }
        return ans;
    }
} flow;
int dinic::dfs(int u,int minn) {
    if(u==t)return minn;
    int ret=0;
    for(int i = carc[u]; minn&&i; i=nxt[i]) {
        carc[u]=i;
        int v = ver[i];
        if(flow[i]&&d[v]==d[u]+1) {
            int final=dfs(v,min(flow[i],minn));
            if(final>0) {
                flow[i]-=final;
                flow[i^1]+=final;
                minn-=final;
```

```
                ret+=final;
            } else d[v]=-1;
        }
    }
    return ret;
}
```

2.1.2 ISAP

```
class ISAP {
    static const int N = 10010;//endpoint_num
    static const int M = 240010;//edge_num
    static const int INF = 0x3f3f3f3f;
    int tot,n,m,s,t;
    int carc[N],gap[N];//curarc and gap
    int pre[N];
    int Head[N],nxt[M],ver[M],flow[M];//base
    int d[N];//depth
    int visit[N];
    bool visited[N];
  public:
    void init(int _n,int _m,int _s,int _t) {
        tot=1;
        n=_n,m=_m,s=_s,t=_t;
        fill(Head,Head+n+1,0);
        fill(visit,visit+n+1,0);
    }
    void addedge(int u,int v,int w) {
        ver[++tot]=v;
        flow[tot]=w;
        nxt[tot]=Head[u];
        Head[u]=tot;

        ver[++tot]=u;
        flow[tot]=0;
        nxt[tot]=Head[v];
        Head[v]=tot;
    }
    bool bfs() {// calculate the depth
        fill(visited,visited+n+1,0);
        queue<int>q;
        visited[t]=1;class ISAP {
    static const int N = 10010;//endpoint_num
    static const int M = 240010;//edge_num
    static const int INF = 0x3f3f3f3f;
    int tot,n,m,s,t;
    int carc[N],gap[N];//curarc and gap
    int pre[N];
    int Head[N],nxt[M],ver[M],flow[M];//base
    int d[N];//depth
    int visit[N];
    bool visited[N];
  public:
    void init(int _n,int _m,int _s,int _t) {
        tot=1;
```

```
        n=_n,m=_m,s=_s,t=_t;
        fill(Head,Head+n+1,0);
        fill(visit,visit+n+1,0);
    }
    void addedge(int u,int v,int w) {
        ver[++tot]=v;
        flow[tot]=w;
        nxt[tot]=Head[u];
        Head[u]=tot;

        ver[++tot]=u;
        flow[tot]=0;
        nxt[tot]=Head[v];
        Head[v]=tot;
    }
    bool bfs() {// calculate the depth
        fill(visited,visited+n+1,0);
        queue<int>q;
        visited[t]=1;
        d[t]=0;
        q.push(t);
        while(q.size()) {-
            int u = q.front();
            q.pop();
            for(int i = Head[u]; i; i=nxt[i]) {
                int v = ver[i];
                if(i&1&&!visited[v]) {
                    visited[v]=true;
                    d[v]=d[u]+1;
                    q.push(v);
                }
            }
        }
        return visited[s];
    }
    int aug() {
        int u=t,df=INF;
        while(u!=s) {// calculate the flow
            df=min(df,flow[pre[u]]);
            u=ver[pre[u]^1];
        }
        u=t;

        while(u!=s) {
            flow[pre[u]]-=df;
            flow[pre[u]^1]+=df;
            u=ver[pre[u]^1];
        }
        return df;
    }
    int maxflow();
} flow;
int ISAP :: maxflow() {
    int ans=0;
    fill(gap,gap+n+1,0);
    for(int i=1; i<=n; i++) carc[i]=Head[i];//copy the head for ignore the
useless edge
    bfs();
```

```
    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.
    int u = s;
    while(d[s]<=n) {
        if(u==t) {
        ans+=aug();
        u=s;
        }
        bool advanced=false;
        for(int i=carc[u]; i; i=nxt[i]) {
            if(flow[i]&&d[u]==d[ver[i]]+1) {
                advanced=true;
                pre[ver[i]]=i;
                carc[u]=i;//carc
                u=ver[i];
                break;
            }
        }
        if(!advanced) {
            int mindep=n-1;
            for(int i=Head[u]; i; i=nxt[i]) {
                if(flow[i]) {
                    mindep=min(mindep,d[ver[i]]);
                }
            }
            if(--gap[d[u]]==0)break;
            gap[d[u]=mindep+1]++;

            carc[u]=Head[u];
            if(u!=s)u=ver[pre[u]^1];
        }
    }
    return ans;
}
        d[t]=0;
        q.push(t);
        while(q.size()) {
            int u = q.front();
            q.pop();
            for(int i = Head[u]; i; i=nxt[i]) {
                int v = ver[i];
                if(i&1&&!visited[v]) {
                    visited[v]=true;
                    d[v]=d[u]+1;
                    q.push(v);
                }
            }
        }
        return visited[s];
    }
    int aug() {
        int u=t,df=INF;
        while(u!=s) {// calculate the flow
            df=min(df,flow[pre[u]]);
            u=ver[pre[u]^1];
        }
        u=t;
```

```
160
161         while(u!=s) {
162             flow[pre[u]]-=df;
163             flow[pre[u]^1]+=df;
164             u=ver[pre[u]^1];
165         }
166         return df;
167     }
168     int maxflow();
169 } flow;
170 int ISAP :: maxflow() {
171     int ans=0;
172     fill(gap,gap+n+1,0);
173     for(int i=1; i<=n; i++) carc[i]=Head[i];//copy the head for ignore the
    useless edge
174     bfs();
175     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.
176     int u = s;
177     while(d[s]<=n) {
178         if(u==t) {
179         ans+=aug();
180         u=s;
181         }
182         bool advanced=false;
183         for(int i=carc[u]; i; i=nxt[i]) {
184             if(flow[i]&&d[u]==d[ver[i]]+1) {
185                 advanced=true;
186                 pre[ver[i]]=i;
187                 carc[u]=i;//carc
188                 u=ver[i];
189                 break;
190             }
191         }
192         if(!advanced) {
193             int mindep=n-1;
194             for(int i=Head[u]; i; i=nxt[i]) {
195                 if(flow[i]) {
196                     mindep=min(mindep,d[ver[i]]);
197                 }
198             }
199             if(--gap[d[u]]==0)break;
200             gap[d[u]=mindep+1]++;
201
202             carc[u]=Head[u];
203             if(u!=s)u=ver[pre[u]^1];
204         }
205     }
206     return ans;
207 }
```

3 Algebraic Algorithms