Fila de Prioridade(HEAP)

A fila de prioridade é uma estrutura de dados que pode ser vista como uma árvore binária praticamente completa. Cada nó da árvore corresponde a um elemento do arranjo que armazena o valor do nó. A árvore está completamente preenchida em todos os níveis, exceto no último nível, que é preenchida a partir da esquerda até certo ponto.

A raiz da árvore é A[1] e, dado o índice i de um nó, os índices do seu PAI(i), do filho da esquerda ESQ(i) e do filho da direita DIR(i) podem ser calculados de modo simples:

```
PAI(i) return i/2
ESQ(i) return 2*i
DIR(i) return 2*i+1
```

Uma fila de prioridade mínima é organizado da seguinte maneira: para todo nó i diferente da raiz ,

```
A[PAI(i)] <= A[i]
```

Como conseqüência imediata dessa organização teremos que A[1] terá o valor mínimo e poderá ser recuperado com 1 operação.

```
MIN() return  A[1]
```

Para obter essa organização um procedimento é necessário: DOWN(i) . DOWN(i) é uma sub-rotina importante para a manutenção de fila de prioridades. Quando DOWN(i) é chamado, supomos que as árvores binárias com raízes em ESQ(i) e DIR(i) são filas de prioridade mínima, mas que A[i] pode ser maior que seus filhos, violando assim a propriedade de fila de prioridade mínima. A função de DOWN(i) é deixar que o valor em A[i] desça na fila de prioridade mínima, de tal forma que a subárvore com raiz no índice i se torne uma fila de prioridade.

```
DOWN(i)
      E  = ESQ(i)
      D  =  DIR(i)
      SE (  E <= TAMANHO_HEAP e A[E] < A[i])
      ENTÃO menor = E
      SENÃO  menor = i
      SE (  D<= TAMANHO_HEAP e A[D] < A[menor])
      ENTÃO menor = D
      SE ( menor != i )
      ENTÃO trocar(A[i],A[menor])
              DOWN(menor)
```

CONSTRUÇÃO FILA DE PRIORIDADE

```
      PARA I:= TAMANHO_HEAP/2 até 1 FAÇA
            DOWN(I)
```

EXTRAIR O MÍNIMO

```
      SE TAMANHO_HEAP < 1 ENTÃO RETURN -1
      MIN = A[1]
      A[1] = A[TAMANHO_HEAP]
      TAMANHO_HEAP--;
      DOWN(1)
      RETURN MIN
```

DECREASE_KEY(i,chave)

```
      A[i] = chave
      ENQUANTO i > 1 e A[PAI(i)] > A[i] FAÇA
       trocar(A[i],A[PAI(i)])
       i = PAI(i)
```

ALGORITMO PRIM

```
#include <values.h>
const int INF = MAXINT/2;

int fixo[MAXN];
int custo[MAXN];

int total = 0;
for(int i=0; i<n; i++) {
  fixo[i] = 0;
  custo[i] = INF;
}
custo[0] = 0;

for(int faltam = n; faltam>0; faltam--) {
  int no = -1;
  for(int i=0; i<n; i++)
    if(!fixo[i] && (no==-1 || custo[i] < custo[no]))
      no = i;
  fixo[no] = 1;

  if(custo[no] == INF) {
    total = INF;
    break;
  }
  total += custo[no];

  for(int i=0; i<n; i++)
    if(custo[i] > G[no][i])
      custo[i] = G[no][i];
}
```

CAMINHO MÍNIMO

```
#include <values.h>
const int INF = MAXINT/2;

int fixo[MAXN];
int dist[MAXN];

for(int i=0; i<n; i++) {
  fixo[i] = 0;
  dist[i] = INF;
}
dist[0] = 0;

for(int faltam = n; faltam>0; faltam--) {
  int no = -1;
  for(int i=0; i<n; i++)
    if(!fixo[i] && (no==-1 || dist[i] < dist[no]))
      no = i;
  fixo[no] = 1;

  if(dist[no] == INF)
    break;

  for(int i=0; i<n; i++)
    if(dist[i] > dist[no]+G[no][i])
      dist[i] = dist[no]+G[no][i];
}
```

CAMINHO MÍNIMO COM HEAP

```
#include <stdio.h>
#define INF 10001
int n;
int i,j;
int g[101][101];
int dist[101];//dist[n] = distancia do vertice inicial até o vertice n
int h[101]; // apontador do heap para o vertice
int hptr[101]; //apontador vertice para heap
int hsize;
int hempty(){
  if( hsize==0 ) return 1;
  return 0;
}
void hup(int p)
{
  int d,ptr;
  ptr = h[p];
  d = dist[h[p]];
  while(p > 1 && d < dist[h[p/2]])
    {
      h[p] = h[p/2];
      hptr[h[p]] = p;
      p /= 2;
    }
  h[p] = ptr;
  hptr[h[p]] = p;
}
void hdown(int p)
{
  int d,ptr;
  ptr = h[p];
  d = dist[h[p]];
  while(2*p <= hsize)
    {
      if(2*p+1 <= hsize && dist[h[2*p]] > dist[h[2*p+1]] &&
        dist[h[2*p+1]] < d)
        {
        h[p] = h[2*p+1];
        hptr[h[p]] = p;
        p = 2*p+1;
        }
      else
      if(dist[h[2*p]] < d)
        {
          h[p] = h[2*p];
          hptr[h[p]] = p;
          p = 2*p;
        }
      else
        break;
    }
  h[p] = ptr;
  hptr[h[p]] = p;
}
```

```
int hpop()
{
  int x;
  x = h[1];
  hptr[x] = -1;
  h[1] = h[hsize--];
  if(hsize != 0)
    hdown(1);
  return x;
}

int main(){

    int a,b,c,u,k,teste=1;

    while( scanf("%d",&n) > 0 &&  n > 0 ){

           for(i=1;i<=n;i++){
             dist[i]=INF;
             h[i]=i;
             hptr[i]=i;
             for(j=1;j<=n;j++) g[i][j]=0;
           }

           while( scanf("%d %d %d",&a,&b,&c) > 0 ){
              if(a+b+c==0)break;
              g[a][b]=g[b][a]=c;
           }

           dist[1]=0;
           hsize = n;
           while( !hempty() ){
              u = hpop();
              if(u==n) break;
              for(k=1;k<=n;k++){
                 if(g[u][k]>0 && dist[k] > dist[u] + g[u][k]){
                     dist[k] = dist[u]+g[u][k];
                     hup(hptr[k]);
                 }

              }
           }
           printf("Teste %d\n",teste++);
           printf("%d\n\n",dist[n]);


    }
}
```

CAMINHO MÍNIMO COM PRIORITY QUEUE C++

```
#include <stdio.h>
#include <queue>

#define MAXN 1001
#define MAXINT MAXN*MAXN
#define range(i,n,m) for(i=n;i<=m;i++)

using namespace std;

int distances[MAXN];
int father[MAXN];
int visit[MAXN];
int g[MAXN][MAXN];
int n,m;

int dijkstra(int start,int end)
{
  priority_queue<pair<int,int> > queue;
  pair <int,int> nodotmp;
  int i, j;

  range(i,1,n){
    distances[i] = MAXINT;
    father[i] = -1;
    visit[i] = false;
  }

  distances[start] = 0;
  queue.push(pair <int,int> (distances[start], start));

  while(!queue.empty()) {
    nodotmp = queue.top();
    queue.pop();
    i = nodotmp.second;
    if (!visit[i]) {
      visit[i] = true;
      range(j,1,n)
        if (!visit[j] && g[i][j] > 0 && distances[i] + g[i][j] < distances[j]) {
          distances[j] = distances[i] + g[i][j];
          father[j] = i;
          queue.push(pair <int,int>(-distances[j], j));
        }
    }
  }
  return distances[end];

}
```

INICIALIZA

```
range(i,1,n) range(j,1,n) g[i][j]=0;
```