1)

```
int setMemberValid(int i){
    if((i<0) || (i>=MAX_MEMBERS)) return 0;
    Return 1
}

int setIsMember(Set*set, int i){
    if(set==NULL) return 0;
    if(!setMemberValid(i)) return 0;
    return((i<<i) & (set->members));

Set* setIntersection(Set*set1, Set*set2){

    int i;
    int minob = (set1->max < set2->max)? set1->max : set2->max;
    int num = (minob-1)/sizeof(int)+1;

    Set* bv = (Set*)malloc(sizeof(Set));
    bv->max = minob;
    bv->vector = (int*)malloc(num*sizeof(int));
    for(i=0; i<num; i++)
        bv->vector[i] = (a->vector[i]) & (b->vector[i]);
    return bv;
}
```

```
void setInsert(Set* set, int i){
    set->vector[i/sizeof(int)] |= 1 << i % sizeof(int);

}
```

2-a)

Retorna o representante que u faz parte.

b) ~~imagine que~~

~~setamos o valor de aux~~

nas linhas de 6 à 8, a partir das variáveis auxiliares x e aux percorremos o caminho da estrutura. "setando" o representante de cada elemento para a raiz.

c)

```
int ub_uniao (UniaoBusca* ub, int u, int v)
    u = ub_busca(ub, u);
    v = ub_busca(ub, v);

    if(u==v) return u;

    if( ub->v[u] > ub->v[v] ){
        ub->v[v] += ub->v[u];
        ub->v[u] = v;
        return v;
    }
    else {
        ub->v[u] += ub->v[v];
        ub->v[v] = u;
        return u;

    }
}
```

3)

a)

nó de origem = 0

corrente = 0

0	0
1	∞
2	∞
3	∞
4	∞
5	∞
6	∞
7	∞
8	∞

visitados = [ ]

naoVisitados = [0,1,2,3,4,5,6,7,8]

0	0
1	6
2	9
3	15
4	~~30~~
5	23
6	∞
7	∞
8	∞

visitados = [0]

naoVisitados = [1,2,3,4,5,6,7,8]

corrente = 1

0	0
1	6
2	9
3	15
4	26
5	23
6	∞
7	∞
8	∞

visitados = [0,1]

naoVisitados = [2,3,4,5,6,7,8]

corrente = 2

0	0
1	6
2	9
3	15
4	26
5	22
6	∞
7	∞
8	37

visitados = [0,1,2]

naoVisitados = [3,4,5,6,7,8]

corrente = 3

0	0
1	6
2	9
3	15
4	19
5	22
6	22
7	∞
8	37

visitados = [0,1,2,3]

naoVisitados = [4,5,6,7,8]

corrente = 5

0	0
1	6
2	9
3	15
4	19
5	22
6	22
7	∞
8	37

visitados = [0,1,2,3,5]

naoVisitados = [4,6,7,8]

corrente = 4

0	0
1	6
2	9
3	15
4	19
5	22
6	22
7	∞
8	37

visitados = [0,1,2,3,4,5]

naoVisitados = [6,7,8]

corrente = 6

0	0
1	6
2	9
3	15
4	19
5	22
6	22
7	37
8	37

visitados = [0,1,2,3,4,5,6]

naoVisitados = [7,8]

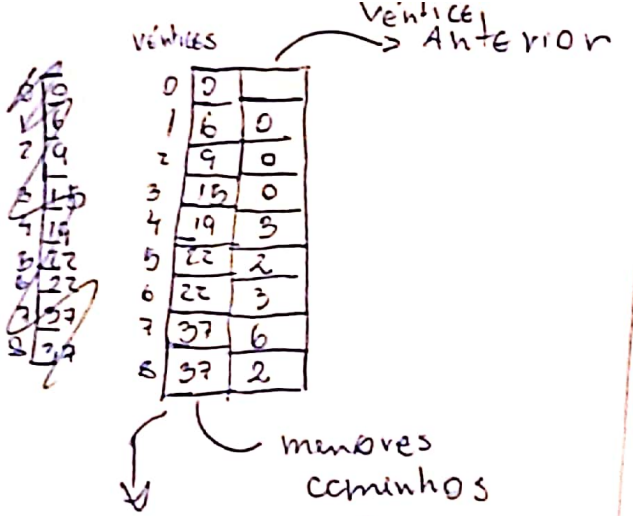
corrente = 7

0	0
1	6
2	9
3	15
4	19
5	22
6	22
7	37
8	37

visitados = [0,1,2,3,4,5,6,7]

naoVisitados = [8]

corrente = 8



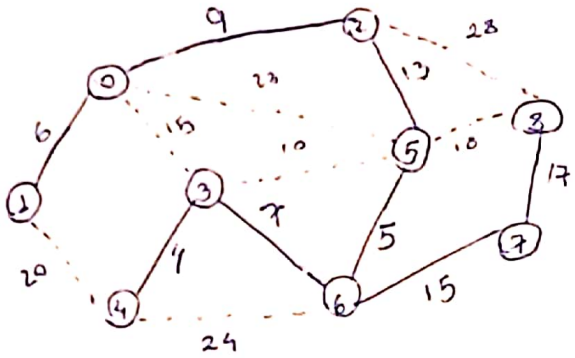
menores caminhos para todos os vértices

Marcos Vinicius Araujo Almeida

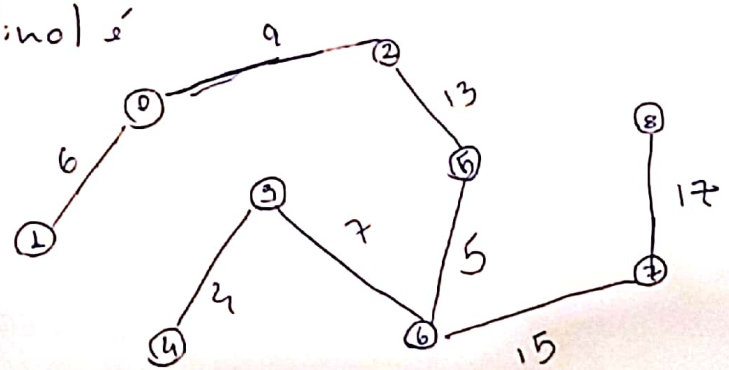
L 191086

C)

1. Devemos considerar cada vértice como uma árvore separada. (Make-Set)
2. Devemos examinar a aresta de menor custo. Se ela une duas árvores na floresta, inclua-a.
3. Repita o passo 2 até todos os nós estarem conectados.



As linhas pontilhadas significam que essa aresta não será incluída na árvore mínima final. Sendo assim, a árvore mínima final é



As linhas pontilhadas não entraram, já que elas fecham um ciclo

4)

```
int temCiclos(grafo *g){

    int* visitado = criaVet(MAX);
    int* pilha = criaPilha(MAX);
    for(int i=0; i<MAX; i++){
        if(cicloAuxiliar(g, i, visitado, pilha))
            return 1;
    }
    return 0;
}

int cicloAuxiliar(grafo *g, int i, int* visitado, int *pilha){
    if(pilha[i])
        return 1;
    if(visitado[i])
        return 0;

    visitado[i] = 1;
    pilha[i] = 1;
    int* listPilhos = g->list[i];
    for(int j
    int n = sizeof(listPilhos) / sizeof(int*);

    for(int j=0; j<n; j++){
        if(cicloAuxiliar(g, listPilhos[j],
            visitado, pilha){
            return 1;
        }
    }

    pilha[i] = 0;

    return 0;
```

Marcos Vinicius Araujo - 1910869