/ /

José Douglas Gondim Soares, 4185417

Avaliação 3 CANA

1- a)

```
Procedimento coberturaIntervalos(Pontos[x1..xn]){
    MergeSort(Pontos[x1..xn]); // Ordena por valor, caso não esteja ordenado.
    intervalos = [];
    Para (i de x1 até xn){
        Se (i.coberto == false){
            addNovoIntervalo(Pontos[x1..xn], intervalos, i);
        }
    }
    retorne intervalos;

Algoritmo addNovoIntervalo(Pontos[x1..xn], intervalos, inicioIntervalo){
    intervalos.addFinal(novo intervalo(inicioIntervalo.valorPonto, inicioIntervalo.valorPonto+1));
    Para(i de inicioIntervalo até xn){
        Se(i.valorPonto < inicioIntervalo.valorPonto+1){
            i.coberto = true;
        }
        break;
    }
}
```

b) Como cada intervalo é iniciado exatamente no começo de um ponto, podemos argumentar que nunca se é desperdiçado espaço à esquerda de qualquer ponto. Isso significa que cada intervalo cobre o máximo de pontos possíveis desde o início ao fim.

CONTINUA...

CONTINUAÇÃO

1- b) Assumindo uma solução S encontrada pelo algoritmo guloso e uma solução T qualquer.

Se $s1 != t1$, isso significa que $t1$ não começa exatamente em um dos pontos e então, espaço à esquerda está sendo desperdiçado. Como consequência, $t1$ pode não cobrir a mesma quantidade de pontos que $s1$. Logo, o algoritmo de T não fez uma escolha ótima naquele momento.

A cada momento que tx é diferente de sx, espaço é desperdiçado, o que leva a provável necessidade de mais intervalos para cobrir os pontos.

Se substituirmos $t1$ por $s1$, veremos que provavelmente cobriremos mais pontos naquele momento e podemos arrastar algumas soluções de T um pouco mais para a direita. O que pode deixar intervalos que não cobrem ponto algum no final.

Conclusão: S sempre será melhor ou igual a T.

1 1

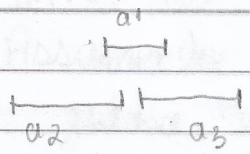
2- a)

Obs: ordem de prioridade sempre ≥ 1. Quanto maior, maior prioridade.

```
Procedimento esperaPrioridade (Programas[1..n]){
    Para (i de 1 até n){
        Programas[i].tempoPonderado = Programas[i].tempo / Programas[i].prioridade;
    }
    Mergesort(Programas); // por ordem de tempoPonderado.
    retorna Programas;
}
```

b) A solução aqui parece extremamente simples, por isso não estou totalmente segura.
Mas o problema da prioridade pode ser resolvido calculando um tempo ponderado da execução de cada programa (tempo/prioridade). Dessa forma, não precisamos mais nos preocupar com a ordem de prioridade, pois tecnicamente, todos os programas agora têm a mesma prioridade.
Logo, a escolha gulosa óbvia é o programa que possui menor tempo ponderado.
Assumindo uma solução S encontrada pela algoritmo e uma solução T qualquer:
Se s_1 é diferente de t_1, logo já podemos assumir que t_1 tem um tempo ponderado maior que s_1. Pois s_1 é o menor.
Sabendo disso, não é difícil ver que se trocar nos t_1 de posição com o programa equivalente a s_1, dentro da solução T, diminuimos o somatório final dos tempos, ou pelo menos ele fica igual. Isso mostra que s_1 é uma escolha gulosa ótima para aquele momento. Fazendo isso várias vezes, podemos concluir $T \geq S$.

3- a)



No contra exemplo acima, o algoritmo selecionaria a atividade a_1 e eliminaria as atividades a_2 e a_3. Mas a solução ótima seria as atividades a_2 e a_3.

c)

```
Procedimento SelAtiv(A[1..n]){
    Se (n=0) retorna (vazio)
    Se (primeira chamada do procedimento){
        a-gul ← atividade com menos conflitos em A
    }Senão{
        a-gul ← atividade que termina mais cedo em A
    }
    A' ← Elimina-conflitos(A, a-gul)
    S ← SelAtividade-gul(A')
    S ← S + a-gul
    retorna(S)
}
```

d) Acabou o tempo, tenho que escanear e mandar, são 17:35

Declaro não trocar informações da prova com colegas e nem pesquisar na internet.

José Douglas Gondim Soares.