1.3 Roteiro de prova

Cortesia da PUC-RJ (adaptado).

300 minutos: INÍCIO DE PROVA

- Abram os seus cadernos de prova e dividam-se para procurar o problema mais fácil: um integrante procura no início, o outro no meio, e o terceiro no fim.
- Quando surgir um problema fácil, todos discutem se ele deve ser o primeiro problema a ser resolvido.
- Quando o primeiro problema for escolhido, quem digita mais rápido o implementa, possivelmente trocando de lugar com quem está no computador.
- Se surgir um problema ainda mais fácil que o primeiro, passe a implementar esse problema.
- Enquanto um integrante resolve o primeiro problema, os outros leem os demais.
- A medida em que os problemas forem lidos (com atenção, sem pular detalhes), preencham a tabela de problemas.
  - AC: recebeu YES (Accepted)? Marque (pinte) a letra do problema na tabela.
  - Ordem: ordem de resolução dos problemas (pode ser infinito).
  - Escrito: se já há código escrito neste problema, mesmo que no papel.
  - Leitores: pessoas que já leram o problema.
  - Complexidade: complexidade da solução implementada.
  - Resumo: resumo sobre o problema.
- Assim que o primeiro problema começar a ser compilado, quem está na frente do PC avisa os outros dois para escolherem o próximo problema mais fácil.
- Assim que o primeiro problema for submetido, mande imprimir o código-fonte e saia do computador.
- Quem for para o computador implementa o segundo problema mais fácil.
- Fora do computador, os outros dois integrantes escolhem a ordem e os resolvedores dos problemas, com base no tempo de implementação.
- Se ninguém tiver alguma ideia para resolver um problema, empurrem-o para o final (ou seja, a ordem desse problema será infinito).
- Quando o segundo problema for submetido (primeiro para avaliação e depois para impressão), saia do computador e reveja a ordenação dos problemas com quem ficou fora do computador.

200 minutos: MEIA-PROVA

- A equipe deve resolver no máximo três problemas ao mesmo tempo.
- Escreva o máximo possível de código no papel.
- Depure com o código do problema e prints de debug (cerr, fprintf(stderr, ...), etc).
  - Explique seu código para outra pessoa da equipe.
  - Acompanhe o código linha por linha, anotando os valores das variáveis e redesenhando as estruturas de dados à medida que forem alteradas.
- Momentos nos quais quem estiver no computador deve avisar os outros membros da equipe:
  - Quando estiver pensando ou debugando.
  - Quando estiver prestes a submeter, para que os outros membros possam fazer testes extras e verificar o formato da saída.
- Logo após submeter, imprima o código.
- Jogue fora as versões mais antigas do código impresso de um programa.
- Jogue fora todos os papéis de um problema quando receber Accepted.
- Mantenha todos os papéis de um problema juntos.

100 minutos: FINAL DE PROVA

- A equipe deve resolver apenas um problema no final da prova.
- Use os balões das outras equipes para escolher o último problema:
  - Os problemas mais resolvidos provavelmente são mais fáceis que os outros problemas.
  - Uma equipe mais bem colocada só é informativa quando as demais não o forem.
- Quem digita mais rápido é quem deve ficar o tempo todo no computador.
- Os outros dois colegas sentam ao lado e ficam dando sugestões para o problema.

60 minutos: PLACAR CONGELADO

- Prestem atenção nos melancias e nas comemorações das outras equipes: os balões continuam vindo (até faltar 30 minutos para o final da prova)

30 minutos: SEM MAIS BALÕES

- Quando terminar um problema, teste com o exemplo de entrada, submeta e só depois pense em mais casos de teste.
- Nos últimos cinco minutos, faça alterações pequenas no código, remova os prints de debug e submeta.

1.4 Bugs do Milênio

Erros teóricos:

- Não ler o enunciado do problema com calma.
- Assumir algum fato sobre a solução na pressa.
- Não reler os limites do problema antes de submeter.
- Quando adaptar um algoritmo, atentar para todos os detalhes da estrutura do algoritmo, se devem (ou não) ser modificados (ex:marcação de vértices/estados).
- O problema pode ser NP, disfarçado ou mesmo sem limites especificados. Nesse caso a solução é bronca mesmo. Não é hora de tentar ganhar o prêmio nobel.

Erros com valor máximo de variável:

- Verificar com calma (fazer as contas direito) para ver se o infinito é tão infinito quanto parece.
- Verificar se operações com infinito estouram 31 bits.
- Usar multiplicação de *int*'s e estourar 32 bits (por exemplo, checar sinais usando $a * b > 0$).

Erros de casos extremos:

- Testou caso $n = 0$? $n = 1$? $n = MAXN$? Muitas vezes tem que tratar separado.
- Pense em todos os casos que podem ser considerados casos extremos ou casos isolados.
- Casos extremos podem atrapalhar não só no algoritmo, mas em coisas como construir alguma estrutura (ex: lista de adj em grafos).
- Não esquecer de self-loops ou multiarestas em grafos.
- Em problemas de caminho Euleriano, verificar se o grafo é conexo.

Erros de desatenção em implementação:

- Errar ctrl-C/ctrl-V em código. Muito comum.
- Colocar igualdade dentro de *if*? ($if(a = 0)continue;$)
- Esquecer de inicializar variável.
- Trocar *break* por *continue* (ou vice-versa).
- Declarar variável global e variável local com mesmo nome (é pedir pra dar merda...).

Erros de implementação:

- Definir variável com tipo errado (*int* por *double*, *int* por *char*).
- Não usar variável com nome *max* e *min*.
- Não esquecer que *.size*() é unsigned.
- Lembrar que 1 é *int*, ou seja, se fizer $long\ long\ a = 1 << 40;$, não irá funcionar (o ideal é fazer $long\ long\ a = 1LL << 40;$).

Erros em limites:

- Qual o ordem do tempo e memória? 10^8 é uma referência para tempo. Sempre verificar rapidamente a memória, apesar de que o limite costuma ser bem grande.
- A constante pode ser muito diminuída com um algoritmo melhor (ex: húngaro no lugar de fluxo) ou com operações mais rápidas (ex: divisões são lentas, bitwise é rápido)?
- O exercício é um caso particular que pode (e está precisando) ser otimizado e não usar direto a biblioteca?

Erros em doubles:

- Primeiro, evitar (a não ser que seja necessário ou mais simples a solução) usar $float/double$. E.g. conta que só precisa de 2 casas decimais pode ser feita com inteiro e depois %100.
- Sempre usar *double*, não $float$ (a não ser que o enunciado peça explicitamente).
- Testar igualdade com tolerância (EPS) — absoluta, e talvez relativa.
- Cuidado com erros de imprecisão, em particular evitar ao máximo subtrair dois números praticamente iguais.

Outros erros:

- Tomar cuidado com possíveis divisões por zero (é *runtime error* na certa).
- Evitar (a não ser que seja necessário) alocação dinâmica de memória.
- Não usar STL desnecessariamente (ex: vector quando um array normal dá na mesma), mas usar se facilitar (ex: nomes associados a vértices de um grafo - $map < string, int >$) ou se precisar (ex: um algoritmo $O(n \log n)$ que usa $< set >$ é necessário para passar no tempo).
- Não inicializar variável a cada teste (zerou vetores? zerou variável que soma algo? zerou com zero? era pra zerar com zero, com -1 ou com INF?).
- Saída está formatada corretamente?
- Declarou vetor com tamanho suficiente?
- Cuidado ao tirar o módulo de número negativo. Ex.: $x\%n$ não dá o resultado esperado se x é negativo, fazer $(x\%n + n)\%n$.