ALGORITMOS E ESTRUTURAS DE DADOS

JOB SELECTION PROBLEM

ALEXANDRE SERRAS (97505) (40%)

JOÃO REIS (98474) (30%)

RICARDO RODRIGUEZ (98388) (30%)

Turma P1

Prof. Tomás Oliveira e Silva

2020/2021

Índice

1. Introdução

1.1 O que é o "generalized weigthed job selection problem"?

Como primeiro trabalho prático de Algoritmos e Estruturas de Dados foi proposto resolver o problema de job selection. Dispomos de um grupo de programadores, designados por P, e um conjunto de Tarefas, caracterizadas pela letra T, que vão ter associadas a estas um valor monetário, uma data de início, uma data de fim e um programador atribuído.

No contexto deste problema, cada tarefa só pode ser desempenhada por um programador e cada programador não pode ter mais do que uma tarefa atribuída no mesmo tempo, isto é, se um programador já tiver uma tarefa atribuída, este só está disponível para lhe atribuírem uma tarefa nova quando acabar a tarefa atual.

Regra geral, o número de tarefas é superior ao número de programadores. Logo, vai acontecer muitas vezes que o total de tarefas que foram realizadas não vai ser igual ao número de tarefas que são dadas.

Como objetivo principal neste problema, queremos descobrir qual é a melhor combinação de tarefas a serem feitas para um determinado grupo de programadores, nas condições referidas anteriormente. A melhor combinação de tarefas será aquela que apresente o maior lucro total, ou seja, aquela cuja soma de todos os lucros das tarefas resulte no maior valor possível entre todas as outras combinações possíveis.

Como objetivos secundários, o grupo implementou soluções para mais alguns problemas, como:

- Indicar o número máximo de tarefas possíveis de se fazerem para P programadores e T tarefas, onde o lucro é totalmente ignorado;
- Indicar, para o caso da melhor combinação de tarefas, quanto dinheiro iria receber cada programador, supondo que o dinheiro era dividido na totalidade pelos programadores que participavam;
- O número de combinações que eram viáveis, isto é, que são possíveis de se realizarem. Desta forma, todas as combinações que não sigam as regras referidas anteriormente, ou seja, todas as que não são válidas, devem ser automaticamente descartadas;
- Criar histogramas dos lucros de todas as soluções viáveis;
- Mostrar, para o caso da melhor combinação possível, o resultado com as tarefas selecionadas e o respetivo programador associado;

2. Abordagem ao problema

2.1. Sucessivos problemas durante a resolução do problema

A primeira abordagem que o grupo teve para resolver o problema foi obter um melhor lucro igual ao que estava apresentado pelo professor nos slides para os parâmetros de entrada 2020 5 2 0, depois de invocar o programa. Antes disto, fomos criar vários casos diferentes para cada combinação possível. Por exemplo, se tivermos o caso '10101', o 1 significa que essa tarefa é realizada enquanto que o 0 significa o caso contrário. Implementámos a solução para o caso '11011' e vimos que o valor do lucro desta combinação correspondia exatamente ao valor que o professor tinha nos slides, confirmando a precisão do programa. Após isto fomos tentar gerar todas as combinações, tal como pensámos inicialmente, e observámos que o valor das combinações ia corresponder aos respetivos valores em binário, que iam sendo incrementados. Por outras palavras, a ideia que o grupo teve foi a de programar um conversor de um número inteiro para um número em binário, de forma a representar uma combinação específica. De seguida, usamos um for loop a gerar todas essas combinações. Por exemplo, se tínhamos 5 tarefas, o ciclo ia gerar combinações de 0 até 2^5 -1, isto é, calcular os números em binário de 0 a 31.

Com esta abordagem, os resultados que obtivemos estavam corretos para alguns casos mais simples. Contudo, para 30 tarefas e 6 programadores, o computador não conseguia correr o programa para estes valores introduzidos. A razão deste problema reside no facto de que a nossa abordagem não estava a gerar as combinações de uma forma recursiva e por estar a tentar guardar todas as combinações que gerávamos, o que não era possível, uma vez que o computador não conseguia guardar tantas combinações a ocupar tanta memória.

Então, implementamos a solução anterior com uma diferença: gerar as combinações de uma maneira recursiva. Usando este método já obtínhamos valores para as 30 tarefas e 6 programadores. Contudo, deparámo-nos com um novo problema, uma vez que o valor do lucro estava bastante mais baixo daquele que era suposto dar.

Então, após alguns testes exaustivos, utilizando vários printf's, vimos que o conversor para binário, que chegamos ao ponto de o alterar ao gerar o número em long long, já não aguentava tal valor, ou seja, estávamos a gerar números maiores do que aqueles que eram permitidos ser gerados, o que provocava overflow.

Nesse momento apercebemo-nos que tínhamos um problema grave que era ter de alterar a forma como gerávamos as combinações.

2.2 Resolução final do problema

Como o problema foi feito sobre um código previamente fornecido pelo professor, vamos apenas explicar as linhas de código que implementamos ou linhas do código que foram alteradas.

```
95
96  typedef struct
97  {
98    int starting_date;      // I starting date of this task
99    int ending_date;        // I ending date of this task
100   int profit;             // I the profit if this task is performed
101   int assigned_to;        // S current programmer number this task is assigned to (use -1 for no assignment)
102 }
103 task_t;
104
105 typedef struct
106 {
107   int NMec;               // I  student number
108   int T;                  // I  number of tasks
109   int P;                  // I  number of programmers
110   int I;                  // I  if 1, ignore profits
111   int total_profit;       // S  current total profit
112   int maxTasks;
113   long best_profit;        // E guardr o melhor lucro
114   long combinacoes;       // E vai guardar o total de combinações
115   int bestCombination [MAX_T] ;// EXTRA vai guardar qual a melhor combinação
116   int dinheiroMedio;      //Extra dinheiro médio por cada programador
117   //fazer um histograma com os lucros
118   double cpu_time;        // S  time it took to find the solution
119   task_t task[MAX_T];     // IS task data
120   int busy[MAX_P];        // S  for each programmer, record until when she/he is busy (-1 means idle)
121   char dir_name[16];      // I  directory name where the solution file will be created
122   char file_name[64];     // I  file name where the solution data will be stored
123 }
```

Figura 1 Estruturas utilizadas

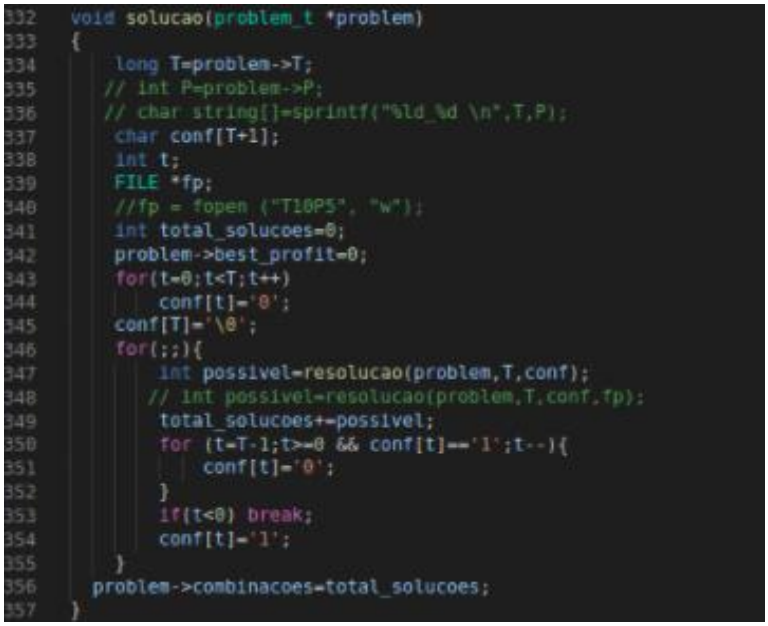
De salientar que na nossa implementação o valor de I é sempre ignorado, pois o I, como se refere aos lucros serem 1 para cada tarefa, vai ser equivalente ao valor do número máximo de tarefas, uma variável que criámos foi a **maxTasks** onde vamos guardar esse valor como um inteiro.

Como se pode observar pela Figura 1, a estrutura referente às tarefas não foi alterada. Contudo, na estrutura referente ao problema, houve algumas alterações que foram feitas. Criamos a variável **best_profit** que guardamos como long para guardar o melhor lucro de todas as combinações para um determinado problema - número mecanográfico, tasks e programadores - e é nesta variável que vamos armazenar a solução para o objetivo principal.

O long **combinacoes** vai corresponder ao número total de combinações que são válidas, ou seja, todas as combinações que se podem realizar, de salientar que cada combinação válida vale 1.

O array de inteiros **bestCombination** vai guardar para a combinação que gera mais lucro, bem como qual programador ficou com determinada tarefa para se obter este resultado.

Por fim, a variável **dinheiroMedio** corresponde ao best_profit a dividir pelo número de programadores, um dado útil para analisarmos um problema que achámos interessante.



```
332  void solucao(problem_t *problem)
333  {
334      long T=problem->T;
335     // int P=problem->P;
336     // char string[]=sprintf("%ld_%d \n",T,P);
337      char conf[T+1];
338      int t;
339      FILE *fp;
340      //fp = fopen ("T10P5", "w");
341      int total_solucoes=0;
342      problem->best_profit=0;
343      for(t=0;t<T;t++)
344          conf[t]='0';
345      conf[T]='\0';
346      for(;;){
347          int possivel=resolucao(problem,T,conf);
348         // int possivel=resolucao(problem,T,conf,fp);
349          total_solucoes+=possivel;
350          for (t=T-1;t>=0 && conf[t]=='1';t--){
351              conf[t]='0';
352          }
353          if(t<0) break;
354          conf[t]='1';
355      }
356    problem->combinacoes=total_solucoes;
357  }
```

Figura 2 Função que gera as combinações

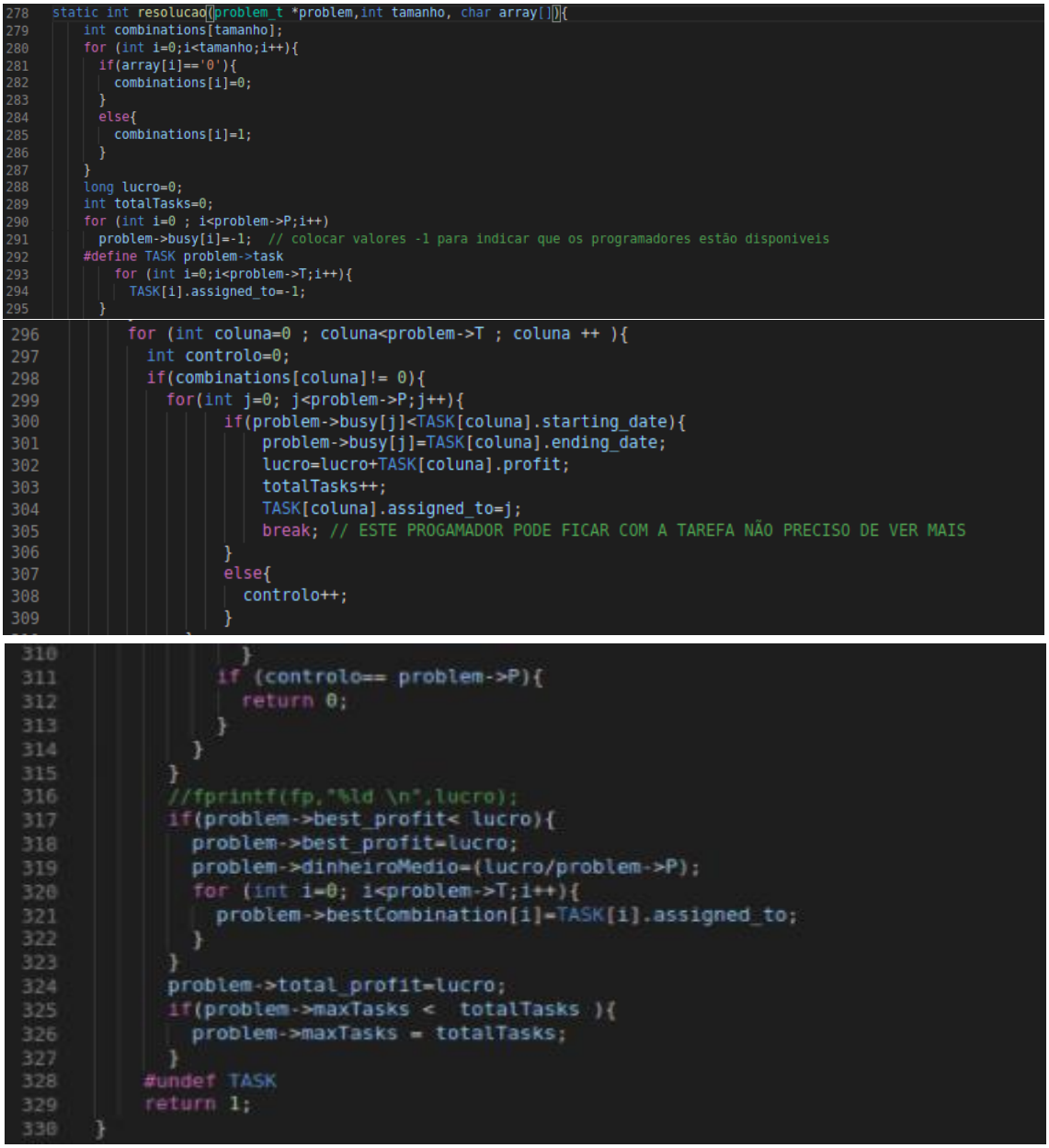
A função **solução** que recebe como argumento apenas um ponteiro para o problema é uma função que vai gerar todas as combinações. Contudo, para não guardar todas as combinações durante o for infinito, o que fazemos é chamar a nossa função resolução que vai resolver o problema.

As linhas que estão em comentário mais a linha 339, são linhas que apenas foram corridas para 2 problemas, pois é com esta que vamos gerar os valores para o histograma. Como não fazia sentido fazer um histograma para todos os problemas, escolhemos que íamos correr estas linhas apenas para 2020 10 5 0 e 2020 20 5 0, quando estes problemas são corridos a linha 347 fica em comentário, pois nestas situações é necessário passar mais um argumento que vai ser um ponteiro para o ficheiro.

A variável **problema->best_profit**=0 serve para garantirmos que no início do problema este é iniciado com o valor a 0, tal como queremos.

A variável int **total_solucoes,** vai ser um contador para o número total de soluções válida e quando chamamos a função resolução esta vai retornar o valor 1 caso seja válida ou 0 no caso de não ser viável. Assim, a soma de todos os retornos vai dar o número total de combinações válidas.

No final do for loop, o valor que está no **total_solucoes** é passado para a variável combinações do problema, para assim ficar armazenada e não ser perdida dentro desta função.



```
static int resolucao(problem_t *problem,int tamanho, char array[]){
    int combinations[tamanho];
    for (int i=0;i<tamanho;i++){
      if(array[i]=='0'){
        combinations[i]=0;
      }
      else{
        combinations[i]=1;
      }
    }
    long lucro=0;
    int totalTasks=0;
    for (int i=0 ; i<problem->P;i++)
      problem->busy[i]=-1;  // colocar valores -1 para indicar que os programadores estão disponiveis
    #define TASK problem->task
        for (int i=0;i<problem->T;i++){
          TASK[i].assigned_to=-1;
      }
        for (int coluna=0 ; coluna<problem->T ; coluna ++ ){
          int controlo=0;
          if(combinations[coluna]!= 0){
            for(int j=0; j<problem->P;j++){
                  if(problem->busy[j]<TASK[coluna].starting_date){
                      problem->busy[j]=TASK[coluna].ending_date;
                      lucro=lucro+TASK[coluna].profit;
                      totalTasks++;
                      TASK[coluna].assigned_to=j;
                      break; // ESTE PROGAMADOR PODE FICAR COM A TAREFA NÃO PRECISO DE VER MAIS
                  }
                  else{
                    controlo++;
                  }
                }
              }
            if (controlo== problem->P){
              return 0;
            }
          }
        }
        //fprintf(fp,"%ld \n",lucro);
        if(problem->best_profit< lucro){
          problem->best_profit=lucro;
          problem->dinheiroMedio=(lucro/problem->P);
          for (int i=0; i<problem->T;i++){
            problem->bestCombination[i]=TASK[i].assigned_to;
          }
        }
        problem->total_profit=lucro;
        if(problem->maxTasks <  totalTasks ){
          problem->maxTasks = totalTasks;
        }
      #undef TASK
      return 1;
    }
```

Figura 3 Função que resolve o problema para cada combinação

Na Figura 3 está representado o código que foi utilizado para resolver o problema, para cada uma das combinações.

A função recebe como argumentos o ponteiro para a estrutura problema_t juntamente com um array de char's que leva 0's e 1's, ou seja, a respetiva combinação e, por fim, recebe o tamanho deste último array.

Por uma questão de simplicidade e para termos de trabalhar com inteiros em vez de caracteres, convertemos o array que nos foi dado como argumento num array de inteiros, **combinations[tamanho].** Visto que apenas tinha 0's e 1's, não existe problema em realizar esta conversão.

Depois, criou-se a variável **lucro** e a variável **totalTasks.** Ambas foram inicializadas com o valor nulo. A variável **lucro** vai corresponder ao lucro total da combinação que está a ser

usada na função enquanto que o **totalTasks** corresponde ao número de tarefas que foram realizadas nesta combinação.

Na structure do problema existe um array chamado busy que tem como tamanho o número máximo de programadores, ou seja 10, contudo nós apenas queremos utilizar consoante o número de programadores do problema, essa parte do vetor e vamos fazer um ciclo for para colocar -1 nessas posições. Com isto vamos garantir que no início do problema todos os programadores estão disponíveis.

Com a mesma ideia, consoante o valor de problema->T, na structure de cada task vamos colocar na mesma o valor -1 inicialmente, ou seja, sabemos que inicialmente a tarefa não está atribuída a ninguém.

Após estes loops iniciais, onde iniciamos os valores, vamos começar a resolver o problema em si, para tal optamos por uma implementação que utiliza 2 ciclos for.

O primeiro vai servir para ir ao vetor **combinations** e verificar a posição do valor que está lá armazenado, sendo que este vai de 0 a problema->T-1 e ainda, entre este loop e o próximo, criamos a variável controlo, que vai servir para validar se esta combinação é viável ou não.

Para cada iteração do loop, caso o valor armazenado no vetor **combinations** seja igual a 1, ou seja diferente de 0, esse valor vai entrar num loop para verificar se este valor pode ser atribuído a um programador, ou se não existe nenhum programador disponível para realizar a tarefa, e aí torna-se não viável a combinação em questão.

Como é que se faz essa verificação? Simplesmente vamos verificar o valor que está no **busy** para um determinado programador. Caso esse valor seja menor do que quando a tarefa em questão acaba (valor este que está guardado em **starting_date** da structure da tarefa), o programador que estávamos a verificar, pode ficar com a tarefa atribuída. Sendo lhe atribuída a tarefa dizemos que o mesmo se encontra indisponível até à data final da tarefa em questão.

O valor do **lucro** é incrementado adicionando-lhe o lucro desta tarefa em questão, o **totalTasks** é incrementado em 1, e o valor da variável **assingned_to** fica com o número do programador que ficou com a tarefa e pode-se dar um break no segundo loop, pois a tarefa já se encontra atribuída a um e apenas um programador.

Caso não se possa entregar a tarefa a um programador é incrementado 1 na variável **controlo.** Caso esta variável fique com o valor igual ao número total de programadores, acontece que, nenhum programador está disponível e aí esta combinação não é valida e retornamos automaticamente o valor 0.

Quando o loop principal não retorna 0, significa que esta combinação é válida e nesses casos, vamos verificar se o valor do **lucro** desta combinação não é maior que o valor do lucro que se encontra atualmente armazenado na variável **problem->best_profit,** em caso afirmativo, atualizamos este valor, alteramos o valor do **dinheiroMedio** por programador e faz-se mais um loop para qual programador ficou com cada tarefa e quais não foram feitas nesta combinação. **É de salientar que este for vai aumentar ainda um tempo significativo o tempo de execução do programa.**

Por fim, antes de fazermos o retorno com o valor 1, significa que a tarefa é viável, verificamos apenas se o valor de **totalTasks**, é superior ao **problema->maxTasks. C**aso seja, este valor é atualizado.

```
380     problem->cpu_time = cpu_time();
381       // call your (recursive?) function to solve the problem here
382       printf("Inicio da tentativa\n");
383
384       solucao(problem);
385       printf("Fim da tentativa\n");
386     problem->cpu_time = cpu_time() - problem->cpu_time;
387     //
388     // save solution data
389     //
390     fprintf(fp,"NMec = %d\n",problem->NMec);
391     fprintf(fp,"T = %d\n",problem->T);
392     fprintf(fp,"P = %d\n",problem->P);
393      fprintf(fp,"Melhor lucro = %ld\n",problem->best_profit);
394       fprintf(fp,"Dinheiro recebido por cada programador = %d\n",problem->dinheiroMedio);
395      fprintf(fp,"Máximo de tasks feitas  = %d\n",problem->maxTasks);
396      fprintf(fp,"Total de soluções viáveis= %ld\n",problem->combinacoes);
397     fprintf(fp,"Solution time = %.3e\n",problem->cpu_time);
398     fprintf(fp,"Task data\n");
399   #define TASK  problem->task[i]
400     fprintf(fp," %s %15s %15s %15s\n","Task number","Starting date","Ending date","Profit");
401     for(i = 0;i < problem->T;i++)
402       fprintf(fp," %d  %15d %15d %22d %3d\n",i,TASK.starting_date,TASK.ending_date,TASK.profit,problem->bestCombination[i]);
403   #undef TASK
```

Figura 4 Chamada da função + valores que são impressos no ficheiro

No final, demonstramos que o programa acabou de correr e guardamos num ficheiro de texto “fp” todos os dados relevantes para análise (melhor combinação, dinheiro recebido por cada programador, etc…) através do fprintf.

Mostrando agora a validação dos dados, vamos apresentar as soluções dos 3 ficheiros de texto que queríamos que tivessem os valores iguais aos que estavam nos slides. É importante salientar mais uma vez que o valor de I deve ser sempre colocado a 0. Caso se coloque I a 1, o valor de melhor lucro possível e o valor máximo de tasks é igual.

```
NMec = 2020
T = 5
P = 2
Melhor lucro = 6505
Dinheiro recebido por cada programador = 3252
Máximo de tasks feitas  = 4
Total de soluções viáveis= 26
Solution time = 6.320e-05
Task data
 Task number   Starting date     Ending date          Profit  Programador
 0                6              12                   1097    0
 1               10              23                   2964    1
 2               11              20                   2048   -1
 3               17              26                   2002    0
 4               24              27                    442    1
End
```

Figura 5 Validação para 2020 5 2

```
NMec = 2020
T = 20
P = 4
Melhor lucro = 36148
Dinheiro recebido por cada programador = 9037
Máximo de tasks feitas  = 12
Total de soluções viáveis= 155510
Solution time = 2.081e-01
Task data
 Task number   Starting date     Ending date          Profit Programa
 0                1               17                  2886   -1
 1                1               17                  4741    0
 2                1               21                  2220   -1
 3                2               13                  1548    1
 4                3               49                  8218    2
 5                5               29                  6188   -1
 6                8               34                  7669    3
 7                9               20                  1483   -1
 8               10               14                   875   -1
 9               14               26                  3400    1
 10              15               30                  3447   -1
 11              20               39                  2016   -1
 12              20               48                  2407   -1
 13              22               25                   251    0
 14              22               38                  3392   -1
 15              24               42                  2397   -1
 16              26               41                  3944    0
 17              28               42                  4228    1
 18              38               48                  1354    3
 19              43               49                   795    0
End
```

Figura 6 Validação 2020 20 4

```
NMec = 2020
T = 30
P = 6
Melhor lucro = 50154
Dinheiro recebido por cada programador = 8359
Máximo de tasks feitas  = 19
Total de soluções viáveis= 208452244
Solution time = 7.958e+01
Task data
```

Task number	Starting date	Ending date	Profit	Programador
0	0	47	5754	-1
1	1	4	437	0
2	1	22	4749	1
3	4	21	3732	2
4	4	49	9216	3
5	5	14	984	-1
6	6	23	4849	0
7	6	44	8750	4
8	9	20	1243	-1
9	11	14	411	-1
10	11	16	1271	5
11	12	24	2564	-1
12	12	25	1823	-1
13	14	25	1987	-1
14	15	27	2167	-1
15	17	30	3036	5
16	22	46	3103	2
17	24	47	4790	0
18	25	36	1151	-1
19	26	34	1350	1
20	26	39	1546	-1
21	29	48	2189	-1
22	29	49	1910	-1
23	30	42	1785	-1
24	31	36	764	5
25	35	38	614	1
26	36	42	992	-1
27	38	46	1984	5
28	39	45	1509	1
29	39	48	673	-1

```
End
```

Figura 7 Validação para 2020 30 6

3. Resolução de outros problemas envolvendo a solução final

O próximo passo foi resolver o problema para cada elemento do grupo e para um número de programadores a variar de 1 a 8 e o número de tarefas de 1 até ao número mais alto que conseguíamos obter, no nosso caso, até 32. Considerando que o número de programadores nunca excede o número de tarefas.

Para tal, criamos um script em bash que compila *job_selection.c* e executa-o com os diferentes argumentos pretendidos. No mesmo script, também retiramos as informações dos ficheiros criados nas sucessivas execuções de job_selection.c, que pretendíamos para posteriormente fazer os gráficos e os histogramas necessários, tais como, o tempo de execução, o número máximo de tarefas realizadas, o melhor lucro e o lucro por cada programador.

3.1. Melhores lucros

Organizamos os melhores lucros por cada estudante na seguinte tabela.

		LUCRO TOTAL		
TAREFAS	**PROGRAMADORES**	**MEC: 97505**	**MEC: 98388**	**MEC: 98474**
1	1	677	2717	1799
2	1	3046	1692	2100
	2	2849	3138	4832
3	1	2123	2980	2917
	2	3284	5741	5859
	3	4725	3894	5273
4	1	5068	6000	3379
	2	6147	3771	5871
	3	5814	7239	12245
	4	9112	10056	6618
5	1	7480	6521	4787
	2	6562	6993	9952
	3	7097	6522	8831
	4	8430	12088	11417
	5	10776	8201	7504
6	1	5831	8298	5199
	2	6917	9194	7997
	3	8545	9014	7544
	4	6647	14858	10855
	5	14130	12000	12467
	6	14163	10900	10869
7	1	9292	5748	7665
	2	10417	6658	8250
	3	7134	9324	9881
	4	9976	9797	8561

	5	16657	10776	12031
	6	12427	14699	15832
	7	10756	15694	10237
8	1	5902	6264	14831
	2	10104	7979	7954
	3	12032	15079	10840
	4	12891	7652	11356
	5	18021	13503	13500
	6	20489	11124	17635
	7	13445	22554	19325
	8	14629	11201	12401
9	1	12298	12058	12370
	2	11007	7351	8144
	3	12594	14090	13386
	4	13536	16429	14552
	5	20633	13730	16533
	6	19267	13946	17639
	7	20786	22596	27377
	8	22041	22815	15780
10	1	10712	11869	15856
	2	12368	14532	16066
	3	11967	12021	11697
	4	19536	14463	10857
	5	21057	19041	20627
	6	13002	14684	18559
	7	15504	15921	17633
	8	25496	20788	30582
11	1	15270	12101	12825
	2	9536	13916	9959
	3	14671	16675	13925
	4	16333	14284	15700
	5	14218	17121	15421
	6	17061	18971	14286
	7	19362	19943	16707
	8	23298	16486	24500
12	1	12053	10933	10920
	2	17404	14107	16424
	3	16525	18110	13966
	4	12543	11072	16926
	5	21458	11449	18138
	6	20675	17881	14969
	7	23945	22127	19898
	8	18524	21923	22581
13	1	12834	12432	29633
	2	13102	17013	16903
	3	13460	16869	14069
	4	18327	13632	20907

	5	20947	18742	14470
	6	22808	18394	24115
	7	23152	31317	19290
	8	20159	26847	30753
14	1	18372	12134	16954
	2	23851	20387	24659
	3	18717	16891	20109
	4	18332	14651	20942
	5	19327	25861	17091
	6	19372	20690	22495
	7	19573	20827	17846
	8	20292	27350	19995
15	1	37080	23604	19835
	2	27940	13632	21248
	3	18113	23168	17177
	4	20550	14199	19923
	5	20800	22040	20340
	6	20605	21687	23731
	7	25270	22522	25679
	8	27397	25512	20698
16	1	18083	16883	28744
	2	14968	17555	24429
	3	21193	21313	21918
	4	19708	19528	20017
	5	20062	22018	19687
	6	22300	20359	20527
	7	26771	26048	26302
	8	25415	20955	26211
17	1	35751	21382	20186
	2	30409	16456	40110
	3	27219	17873	22294
	4	19048	28207	18689
	5	21237	23275	22236
	6	29737	26374	26399
	7	23248	25959	29562
	8	28449	29034	29051
18	1	23567	26776	15616
	2	20530	22644	21742
	3	25332	25073	27961
	4	21181	20658	20686
	5	23018	19222	27669
	6	24913	23693	25956
	7	19790	23676	27071
	8	26769	31709	25086
19	1	21865	33957	28323
	2	26515	34809	23371
	3	30436	28913	25522

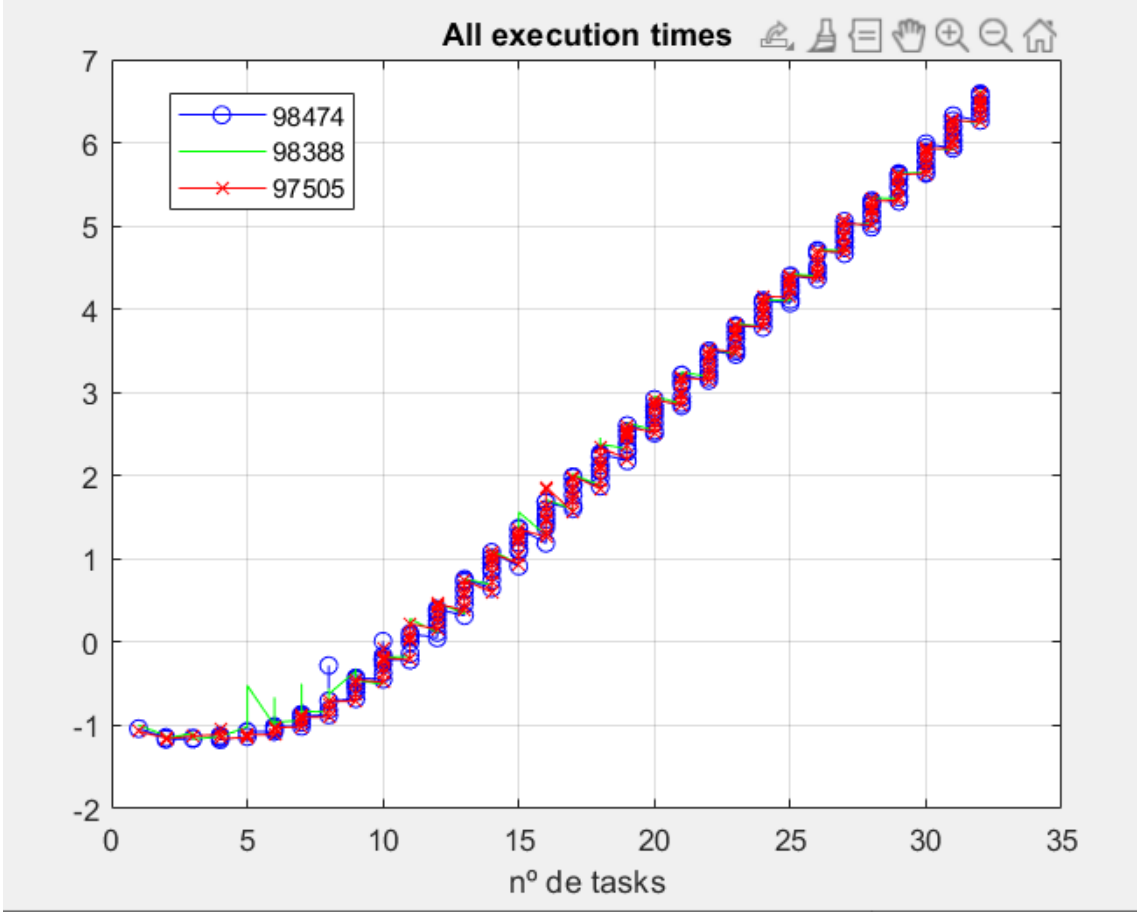
	4	24924	21960	30339
	5	26067	22338	22458
	6	20959	28257	30076
	7	23144	28743	31067
	8	30702	25270	22970
20	1	28683	24214	23744
	2	24248	48010	21247
	3	25491	25346	28114
	4	31531	27145	28434
	5	32855	27984	26743
	6	27926	26231	25749
	7	26157	26838	27549
	8	30522	28688	35420
21	1	26547	27304	30099
	2	28996	17699	39035
	3	25907	31376	32456
	4	27613	27238	26071
	5	35385	25000	27359
	6	30076	37130	29687
	7	21593	24765	26691
	8	36299	31992	34594
22	1	33902	20235	41939
	2	22828	27704	37328
	3	28724	36156	34165
	4	23236	36120	28821
	5	27870	24393	28728
	6	31405	30323	29509
	7	29887	30376	34240
	8	30353	33818	28425
23	1	16504	34853	35163
	2	33516	28134	25627
	3	32539	33067	30539
	4	34581	32925	23588
	5	33156	34248	31632
	6	32681	36075	24072
	7	28614	29191	24436
	8	30042	24747	27348
24	1	34628	42216	26505
	2	24965	46340	37139
	3	38604	30929	30146
	4	31600	35991	39106
	5	30825	29305	32855
	6	30172	31869	26931
	7	30095	30777	28027
	8	30116	37315	42552
25	1	33179	37422	28518
	2	29572	34604	39184

	3	40805	32144	38613
	4	34272	38939	36430
	5	28912	29572	30361
	6	32079	36045	31611
	7	41010	37457	31070
	8	32635	34120	34969
26	1	50167	29670	41762
	2	28245	33268	42567
	3	36033	41627	37294
	4	33345	39835	33047
	5	39361	38157	37937
	6	38266	40180	37883
	7	26453	41128	34047
	8	29194	36115	34533
27	1	41619	25042	70628
	2	33937	34213	31040
	3	40116	33699	39495
	4	38386	38348	42247
	5	33843	41659	38444
	6	36535	31986	28902
	7	42535	40820	43104
	8	34487	31092	32743
28	1	73541	28839	49548
	2	36838	36552	54947
	3	42038	55549	49525
	4	40573	35722	40646
	5	40575	42159	31898
	6	45341	40069	38304
	7	38698	36128	28967
	8	36458	28825	36515
29	1	25919	29761	36833
	2	46237	53179	39601
	3	38056	46996	38680
	4	42227	53809	36517
	5	42912	45895	38586
	6	41421	37307	42044
	7	35995	39862	46132
	8	32116	37850	41052
30	1	30811	39387	34457
	2	36923	37435	47801
	3	41615	53736	44321
	4	39206	41793	47563
	5	45262	46150	54131
	6	41881	46711	46322
	7	34175	43787	34529
	8	39956	39586	47014
31	1	28777	64493	36078

	2	37412	29587	48310
	3	50669	61522	42122
	4	43839	42140	48870
	5	45719	41445	38821
	6	45265	44650	54079
	7	47217	45827	44565
	8	33416	42411	40707
32	1	40594	55403	23133
	2	32734	62611	52839
	3	37065	43648	37249
	4	45788	42332	38958
	5	48405	45821	47867
	6	35849	45589	43429
	7	40731	43210	43728
	8	44579	45909	46168

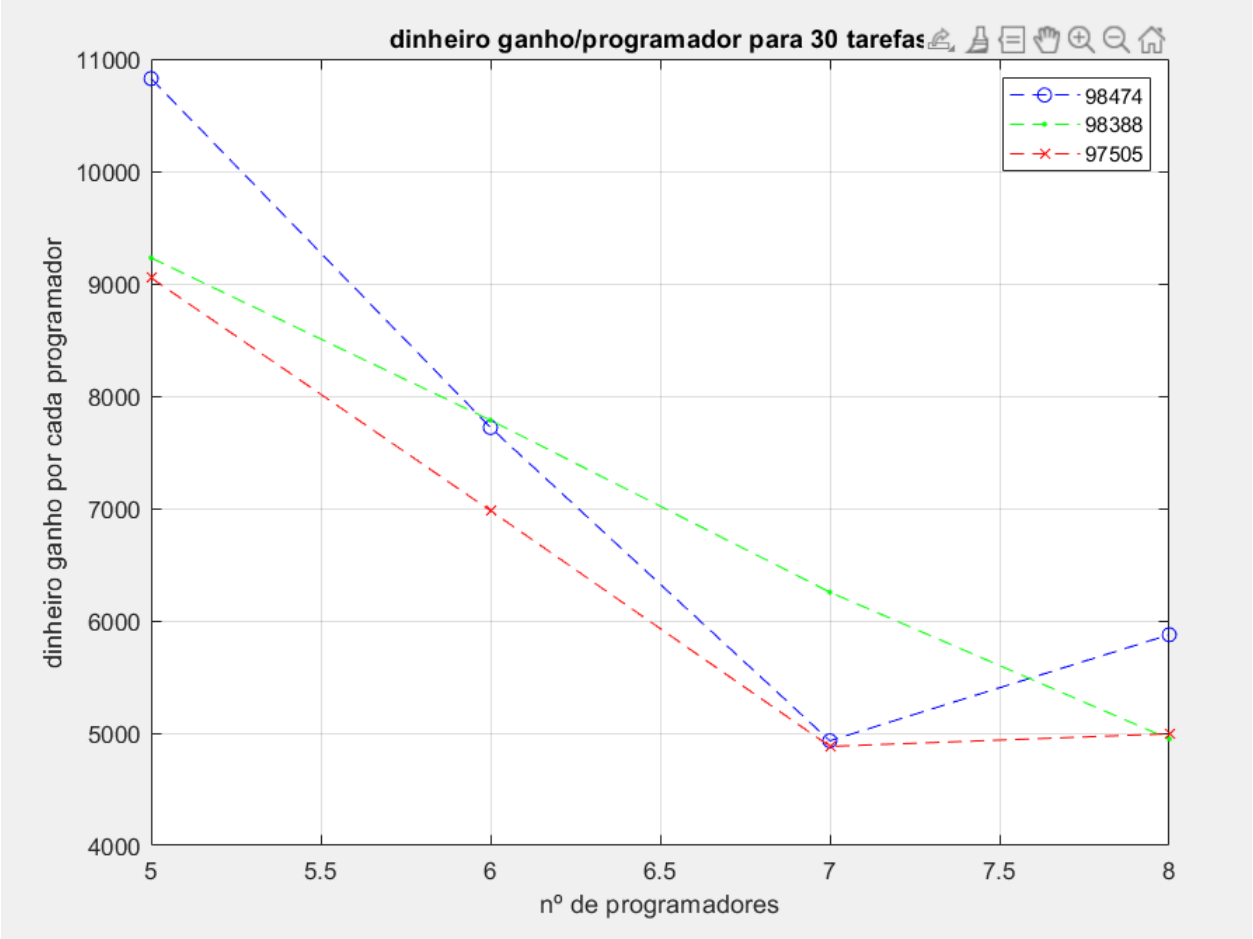
3.2. Tempos de execução

O seguinte gráfico, desenhado recorrendo ao Matlab, reflete todos os tempos de execução de cada aluno.



Concluímos então, por análise do gráfico, que o tempo de execução aumenta de forma exponencial á medida que o número de tarefas, e por sua vez, o número de programadores, aumentam, o que seria de esperar.

3.3. Lucro por programador



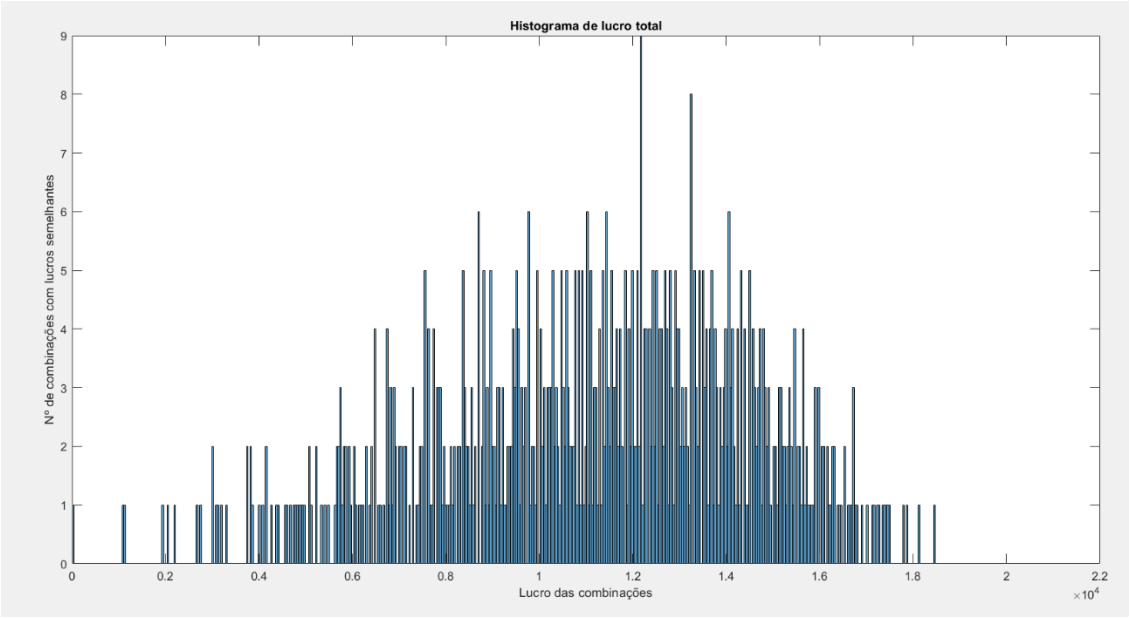
Este gráfico traduz o dinheiro ganho por programador em 30 tarefas. Como se verifica, têm, em geral, uma forma aproximadamente decrescente, ou seja, à medida que o número de programadores aumenta para as mesmas tarefas (30), o dinheiro ganho por cada programador vai diminuindo.

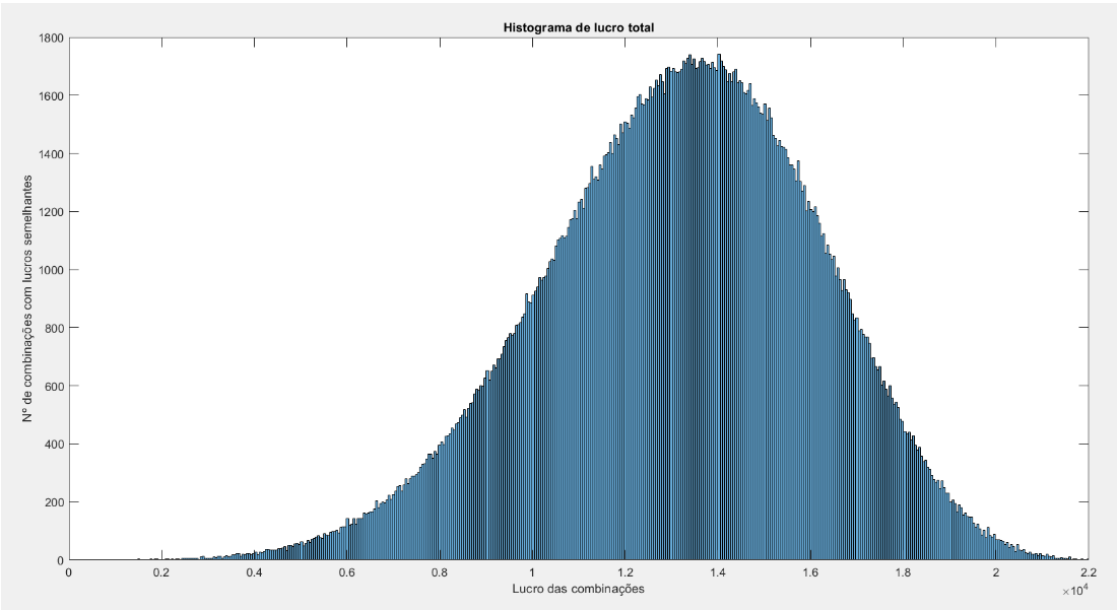
Porém, o número mecanográfico 98474 e, ligeiramente, o número mecanográfico 97505, sofrem um aumento de lucro por programador quando as 30 tarefas são realizadas por 8 programadores uma vez que, nesse caso, o melhor lucro aumentou significativamente em relação ao melhor lucro das realizadas por número de programadores menores.

3.4. Distribuição de frequências de lucros das combinações

Achámos importante analisar a forma como os lucros de todas as combinações possíveis estão distribuídas e, para isso, usámos o Matlab para poder criar um histograma a partir dos dados que recebemos do programa principal (que estão delimitados como comentário).

Assim, seguem-se dois histogramas. O primeiro apresenta os valores de quando T = 10 e P = 5, enquanto que o segundo apresenta os valores de quando T = 20 e P = 5.





Observando agora o gráfico, verifica-se que o primeiro histograma apresenta um número de combinações com lucros semelhantes bastante instáveis entre a vizinhança, um determinado lucro pode ter um valor baixo e, de seguida, o valor de lucro seguinte pode ter um número alto de combinações associadas a esta. Isto acontece uma vez que o número de

combinações para T = 10 é um número relativamente pequeno quando introduzimos o segundo parâmetro da função histogram() com um valor de nBins relativamente elevado (500).

No entanto, analisando agora o segundo histograma, verifica-se que o gráfico segue uma distribuição normal, ou Gaussiana, bastante bem definida e com uma diferença quase mínima de número de combinações por lucro entre colunas próximas, ao contrário do gráfico do histograma 1. Isto acontece uma vez que o histograma 2 é usado para quando T = 20, ou seja, o número de combinações diferentes geradas é bastante elevado para permitir apresentar um gráfico praticamente contínuo.

4. Conclusão

Os resultados deste trabalho foram os esperados, uma vez que, ao nível dos lucros, era esperado que quanto mais programadores houvessem maior iria ser o resultado; ao nível dos tempos de execução, era esperado que a cada tarefa, o tempo de execução aumenta-se para o dobro das combinações possíveis, e quanto mais programadores existem, maior é o tempo de execução, pois mais verificações é necessário fazer-se em cada ciclo for.

Em geral, gostamos muito de estar envolvidos neste projeto uma vez que nos familiarizou ao nível da linguagem C, bem como do Matlab que foi preciso para desenvolver os gráficos, ao nível do trabalho coletivo e, o mais importante, enriqueceu o nosso raciocínio para enfrentar problemas que precisam de uma solução algorítmica complexa, como é o caso deste problema que nos foi dado.

5. Anexos

Código de “job_selection.c” em linguagem C:

```c
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// AED, 2020/2021
//
// Alexandre Serras 97505 (40%)
// João Reis 98474  (30%)
// Ricardo Rodriguez 98388  (30%)
//
// Brute-force solution of the generalized weighted job selection problem
//
// Compile with "cc -Wall -O2 job_selection.c -lm" or equivalent
//
// In the generalized weighted job selection problem we will solve here we have T programming tasks and P programmers.
// Each programming task has a starting date (an integer), an ending date (another integer), and a profit (yet another
// integer). Each programming task can be either left undone or it can be done by a single programmer. At any given
// date each programmer can be either idle or it can be working on a programming task. The goal is to select the
// programming tasks that generate the largest profit.
//
// Things to do:
//   0. (mandatory)
//      Place the student numbers and names at the top of this file.
//   1. (highly recommended)
//      Read and understand this code.
//   2. (mandatory)
//      Solve the problem for each student number of the group and for
//        N=1, 2, ..., as higher as you can get and
//        P=1, 2, ... min(8,N)
//      Present the best profits in a table (one table per student number).
//      Present all execution times in a graph (use a different color for the times of each student number).
//      Draw the solutions for the highest N you were able to do.
//   3. (optional)
//      Ignore the profits (or, what is the same, make all profits equal); what is the largest number of programming
//      tasks that can be done?
//   4. (optional)
//      Count the number of valid task assignments. Calculate and display an histogram of the number of occurrences of
//      each total profit. Does it follow approximately a normal distribution?
//   5. (optional)
//      Try to improve the execution time of the program (use the branch-and-bound technique).
//      Can you use divide and conquer to solve this problem?
//      Can you use dynamic programming to solve this problem?
//   6. (optional)
//      For each problem size, and each student number of the group, generate one million (or more!) valid random
//      assignments and compute the best solution found in this way. Compare these solutions with the ones found in
//      item 2.
//   7. (optional)
//      Surprise us, by doing something more!
//   8. (mandatory)
//      Write a report explaining what you did. Do not forget to put all your code in an appendix.
//

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include "../P02/elapsed_time.h"


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// Random number generator interface (do not change anything in this code section)
//
// In order to ensure reproducible results on Windows and GNU/Linux, we use a good random number generator, available at
//   https://www-cs-faculty.stanford.edu/~knuth/programs/rng.c
// This file has to be used without any modifications, so we take care of the main function that is there by applying
// some C preprocessor tricks
//

#define main  rng_main                          // main gets replaced by rng_main
#ifdef __GNUC__
int rng_main() __attribute__((__unused__));   // gcc will not complain if rnd_main() is not used
#endif
#include "rng.c"
#undef main                                     // main becomes main again

#define srandom(seed)  ran_start((long)seed)  // start the pseudo-random number generator
#define random()       ran_arr_next()         // get the next pseudo-random number (0 to 2^30-1)


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// problem data (if necessary, add new data fields in the structures; do not change anything else in this code section)
//
// on the data structures declared below, a comment starting with
// * a I means that the corresponding field is initialized by init_problem()
// * a S means that the corresponding field should be used when trying all possible cases
// * IS means both (part initialized, part used)
//


#if 1

#define MAX_T  64  // maximum number of programming tasks
#define MAX_P  10  // maximum number of programmers

typedef struct
```

```c
{
  int starting_date;      // I starting date of this task
  int ending_date;        // I ending date of this task
  int profit;             // I the profit if this task is performed
  int assigned_to;        // S current programmer number this task is assigned to (use -1 for no assignment)
}
task_t;

typedef struct
{
  int NMec;               // I  student number
  int T;                  // I  number of tasks
  int P;                  // I  number of programmers
  int I;                  // I  if 1, ignore profits
  int total_profit;       // S  current total profit
  int maxTasks;
  long best_profit;        // E guardr o melhor lucro
  long combinacoes;       // E vai guardar o total de combinações
  int bestCombination [MAX_T] ;// EXTRA vai guardar qual a melhor combinação
  int dinheiroMedio;      //Extra dinheiro médio por cada programador
  //fazer um histograma com os lucros
  double cpu_time;        // S  time it took to find the solution
  task_t task[MAX_T];     // IS task data
  int busy[MAX_P];        // S  for each programmer, record until when she/he is busy (-1 means idle)
  char dir_name[16];      // I  directory name where the solution file will be created
  char file_name[64];     // I  file name where the solution data will be stored
}
problem_t;

int compare_tasks(const void *t1,const void *t2)
{
  int d1,d2;

  d1 = ((task_t *)t1)->starting_date;
  d2 = ((task_t *)t2)->starting_date;
  if(d1 != d2)
    return (d1 < d2) ? -1 : +1;
  d1 = ((task_t *)t1)->ending_date;
  d2 = ((task_t *)t2)->ending_date;
  if(d1 != d2)
    return (d1 < d2) ? -1 : +1;
  return 0;
}

void init_problem(int NMec,int T,int P,int ignore_profit,problem_t *problem)
{
  int i,r,scale,span,total_span;
  int *weight;

  //
  // input validation
  //
  if(NMec < 1 || NMec > 999999)
  {
    fprintf(stderr,"Bad NMec (1 <= NMex (%d) <= 999999)\n",NMec);
    exit(1);
  }
  if(T < 1 || T > MAX_T)
  {
    fprintf(stderr,"Bad T (1 <= T (%d) <= %d)\n",T,MAX_T);
    exit(1);
  }
  if(P < 1 || P > MAX_P)
  {
    fprintf(stderr,"Bad P (1 <= P (%d) <= %d)\n",P,MAX_P);
    exit(1);
  }
  //
  // the starting and ending dates of each task satisfy 0 <= starting_date <= ending_date <= total_span
  //
  total_span = (10 * T + P - 1) / P;
  if(total_span < 30)
    total_span = 30;
  //
  // probability of each possible task duration
  //
  // task span relative probabilities
```

```
  //
  // |  0  0  4  6  8 10 12 14 16 18 | 20 | 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1 | smaller than 1
  // |  0  0  2  3  4  5  6  7  8  9 | 10 | 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 | 30 31 ... span
  //
  weight = (int *)alloca((size_t)(total_span + 1) * sizeof(int)); // allocate memory (freed automatically)
  if(weight == NULL)
  {
    fprintf(stderr,"Strange! Unable to allocate memory\n");
    exit(1);
  }
#define sum1  (298.0)                     // sum of weight[i] for i=2,...,29 using the data given in the comment above
#define sum2  ((double)(total_span - 29))  // sum of weight[i] for i=30,...,data_span using a weight of 1
#define tail  100
  scale = (int)ceil((double)tail * 10.0 * sum2 / sum1); // we want that scale*sum1 >= 10*tail*sum2, so that large task
  if(scale < tail)                                      // durations occur 10% of the time
    scale = tail;
  weight[0] = 0;
  weight[1] = 0;
  for(i = 2;i <= 10;i++)
    weight[i] = scale * (2 * i);
  for(i = 11;i <= 29;i++)
    weight[i] = scale * (30 - i);
  for(i = 30;i <= total_span;i++)
    weight[i] = tail;
#undef sum1
#undef sum2
#undef tail
  //
  // accumulate the weigths (cummulative distribution)
  //
  for(i = 1;i <= total_span;i++)
    weight[i] += weight[i - 1];
  //
  // generate the random tasks
  //
  srandom(NMec + 314161 * T + 271829 * P);
  problem->NMec = NMec;
  problem->T = T;
  problem->P = P;
  problem->I = (ignore_profit == 0) ? 0 : 1;
  for(i = 0;i < T;i++)
  {
    //
    // task starting an ending dates
    //
    r = 1 + (int)random() % weight[total_span]; // 1 .. weight[total_span]
    for(span = 0;span < total_span;span++)
      if(r <= weight[span])
        break;
    problem->task[i].starting_date = (int)random() % (total_span - span + 1);
    problem->task[i].ending_date = problem->task[i].starting_date + span - 1;
    //
    // task profit
    //
    // the task profit is given by r*task_span, where r is a random variable in the range 50..300 with a probability
    //   density function with shape (two triangles, the area of the second is 4 times the area of the first)
    //
    //      *
    //     /|   *
    //    / |       *
    //   /  |           *
    //  *---*--------------*
    // 50 100 150 200 250 300
    //
    scale = (int)random() % 12501; // almost uniformly distributed in 0..12500
    if(scale <= 2500)
      problem->task[i].profit = 1 + round((double)span * (50.0 + sqrt((double)scale)));
    else
      problem->task[i].profit = 1 + round((double)span * (300.0 - 2.0 * sqrt((double)(12500 - scale))));
  }
  //
  // sort the tasks by the starting date
  //
  qsort((void *)&problem->task[0],(size_t)problem->T,sizeof(problem->task[0]),compare_tasks);
  //
  // finish
  //
  if(problem->I != 0)
    for(i = 0;i < problem->T;i++)
      problem->task[i].profit = 1;
#define DIR_NAME  problem->dir_name
  if(snprintf(DIR_NAME,sizeof(DIR_NAME),"%06d",NMec) >= sizeof(DIR_NAME))
  {
    fprintf(stderr,"Directory name too large!\n");
```

```
    exit(1);
  }
#undef DIR_NAME
#define FILE_NAME  problem->file_name
  if(snprintf(FILE_NAME,sizeof(FILE_NAME),"%06d/%02d_%02d_%d.txt",NMec,T,P,problem->I) >= sizeof(FILE_NAME))
  {
    fprintf(stderr,"File name too large!\n");
    exit(1);
  }
#undef FILE_NAME
}

#endif


////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// problem solution (place your solution here)
//

static int resolucao(problem_t *problem,int tamanho, char array[]){
    int combinations[tamanho];
    for (int i=0;i<tamanho;i++){
      if(array[i]=='0'){
        combinations[i]=0;
      }
      else{
        combinations[i]=1;
      }
    }
    long lucro=0;
    int totalTasks=0;
    for (int i=0 ; i<problem->P;i++)
      problem->busy[i]=-1;  // colocar valores -1 para indicar que os programadores estão disponiveis
    #define TASK problem->task
        for (int i=0;i<problem->T;i++){
          TASK[i].assigned_to=-1;
      }
      for (int coluna=0 ; coluna<problem->T ; coluna ++ ){
        int controlo=0;
        if(combinations[coluna]!= 0){
          for(int j=0; j<problem->P;j++){
                if(problem->busy[j]<TASK[coluna].starting_date){
                    problem->busy[j]=TASK[coluna].ending_date;
                    lucro=lucro+TASK[coluna].profit;
                    totalTasks++;
                    TASK[coluna].assigned_to=j;
                    break; // ESTE PROGAMADOR PODE FICAR COM A TAREFA NÃO PRECISO DE VER MAIS
                }
                else{
                  controlo++;
                }
            }
          if (controlo== problem->P){
            return 0;
          }
        }
      }
      //fprintf(fp,"%ld \n",lucro);
      if(problem->best_profit< lucro){
        problem->best_profit=lucro;
        problem->dinheiroMedio=(lucro/problem->P);
        for (int i=0; i<problem->T;i++){
          problem->bestCombination[i]=TASK[i].assigned_to;
        }
      }
      problem->total_profit=lucro;
      if(problem->maxTasks <  totalTasks ){
        problem->maxTasks = totalTasks;
      }
    #undef TASK
    return 1;
}

void solucao(problem_t *problem)
{
    long T=problem->T;
   // int P=problem->P;
   // char string[]=sprintf("%ld_%d \n",T,P);
    char conf[T+1];
    int t;
    FILE *fp;
    //fp = fopen ("T10P5", "w");
    int total_solucoes=0;
    problem->best_profit=0;
    for(t=0;t<T;t++)
        conf[t]='0';
    conf[T]='\0';
    for(;;){
        int possivel=resolucao(problem,T,conf);
       // int possivel=resolucao(problem,T,conf,fp);
        total_solucoes+=possivel;
        for (t=T-1;t>=0 && conf[t]=='1';t--){
            conf[t]='0';
```

```
            }
            if(t<0) break;
            conf[t]='1';
        }
    problem->combinacoes=total_solucoes;
}


#if 1

static void solve(problem_t *problem)
{
  FILE *fp;
  int i;

  //
  // open log file
  //
  (void)mkdir(problem->dir_name,S_IRUSR | S_IWUSR | S_IXUSR);
  fp = fopen(problem->file_name,"w");
  if(fp == NULL)
  {
    fprintf(stderr,"Unable to create file %s (maybe it already exists? If so, delete it!)\n",problem->file_name);
    exit(1);
  }
  //
  // solve
  //
  problem->cpu_time = cpu_time();
    // call your (recursive?) function to solve the problem here
    printf("Inicio da tentativa\n");

    solucao(problem);
    printf("Fim da tentativa\n");
  problem->cpu_time = cpu_time() - problem->cpu_time;
  //
  // save solution data
  //
  fprintf(fp,"NMec = %d\n",problem->NMec);
  fprintf(fp,"T = %d\n",problem->T);
  fprintf(fp,"P = %d\n",problem->P);
   fprintf(fp,"Melhor lucro = %ld\n",problem->best_profit);
    fprintf(fp,"Dinheiro recebido por cada programador = %d\n",problem->dinheiroMedio);
   fprintf(fp,"Máximo de tasks feitas  = %d\n",problem->maxTasks);
   fprintf(fp,"Total de soluções viáveis= %ld\n",problem->combinacoes);
  fprintf(fp,"Solution time = %.3e\n",problem->cpu_time);
  fprintf(fp,"Task data\n");
#define TASK  problem->task[i]
  fprintf(fp," %s %15s %15s %15s\n","Task number","Starting date","Ending date","Profit");
  for(i = 0;i < problem->T;i++)
    fprintf(fp," %d  %15d %15d %22d %3d\n",i,TASK.starting_date,TASK.ending_date,TASK.profit,problem->bestCombination[i]);
#undef TASK

  fprintf(fp,"End\n");
  //
  // terminate
  //
  if(fflush(fp) != 0 || ferror(fp) != 0 || fclose(fp) != 0)
  {
    fprintf(stderr,"Error while writing data to file %s\n",problem->file_name);
    exit(1);
  }
}

#endif



////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// main program
//

int main(int argc,char **argv)
{
  problem_t problem;
  int NMec,T,P,I;

  NMec = (argc < 2) ? 2020 : atoi(argv[1]);
  T = (argc < 3) ? 5 : atoi(argv[2]);
  P = (argc < 4) ? 2 : atoi(argv[3]);
  I = (argc < 5) ? 0 : atoi(argv[4]);
  init_problem(NMec,T,P,I,&problem);
  solve(&problem);
  return 0;
}
```

Código em bash para executar “job_selection.c” com vários argumetos pretendidos:

```bash
#!/bin/bash

declare -a Nmec=(97505 98474 98388)

declare -a tasks=( $(seq 1 32) )
declare -a programers=( $(seq 1 8) )

gcc -Wall -O2 job_selection.c -o a -lm


for s in ${Nmec[@]}; do

    for t in ${tasks[@]}; do
        for p in ${programers[@]}; do
            if (( $t >= $p )); then
                (./a $s $t $p 0)
            fi
        done
    done

    cd 0$s
    grep "Melhor lucro = " *.txt | sed -e 's/_0.txt:Melhor lucro =//' -e 's/_/ /' >lucro_$s
    grep "Solution time = " *.txt | sed -e 's/_0.txt:Solution time =//' -e 's/_/ /' >STime_$s
    grep "Máximo de tasks feitas  =" *.txt | sed -e 's/_0.txt:Máximo de tasks feitas  =//' -e 's/_/ /' >MaxT_$s
    grep "Dinheiro recebido por cada programador =" *.txt | sed -e 's/_0.txt:Dinheiro recebido por cada programador =//' -e 's/_/ /' >LP_$s
    cd ..
done
```

Código no Matlab para o desenho do gráfico “All execution times”:

```matlab
load STime_98474.txt;
load STime_98388.txt;
load STime_97505.txt;
j=STime_98474;
r=STime_98388;
a=STime_97505;

figure(1);
plot(j(:,1),log10(j(:,3)*10e3),'bo-', r(:,1),log10(r(:,3)*10e3),'g-', a(:,1),log10(a(:,3)*10e3),'rx-');
grid on;
xlabel('n° de tasks');
legend('98474','98388','97505');
title('All execution times');
```

Código no Matlab para o desenho do gráfico do lucro por programador:

```matlab
load LP_98474.txt;
load LP_98388.txt;
load LP_97505.txt;

j=LP_98474;
r=LP_98388;
a=LP_97505;

figure(2);
plot(j(:,1),j(:,2),'bo--', r(:,1),r(:,2),'g.--', a(:,1),a(:,2),'rx--');
grid on;
xlabel('n° de programadores');
ylabel('dinheiro ganho por cada programador');
legend('98474','98388','97505');
title('dinheiro ganho/programador para 30 tarefas');
```

Código no Matlab para a construção dos histogramas:

```
file = fopen("T10_P5.txt",'r');
dados = textscan(file,"%d");
fclose(file);
dados = dados{1,1};

histogram(dados,500);
xlabel("Lucro das combinações");
ylabel("N° de combinações com lucros semelhantes");
title("Histograma de lucro total");
xlim([0 22000]);
```

```
file = fopen("T20P5.txt",'r');
dados = textscan(file,"%d");
fclose(file);
dados = dados{1,1};

histogram(dados,500);
xlabel("Lucro das combinações");
ylabel("N° de combinações com lucros semelhantes");
title("Histograma de lucro total");
xlim([0 22000]);
```