XLI OLIMPIADA WIEDZY TECHNICZNEJ

Zawody II stopnia

Rozwiązania zadań dla grupy mechaniczno-budowlanej

Rozwiązanie zadania 1

Spełnienie warunku b)

Maksymalny moment zginający występuje w przekroju utwierdzenia wspornika i jest równy:

$$M_{max} = P\,l\,. \tag{1}$$

Maksymalne naprężenia od zginania wspornika występuje także w przekroju utwierdzenia i są równe:

$$\sigma = \frac{M_{max}}{W} \leq k\,. \tag{2}$$

Wskaźnik wytrzymałości W przekroju prostokątnego, jest równy:

$$W = \frac{b\,h^2}{6}\,. \tag{3}$$

Po odpowiednich przekształceniach, otrzymuje się z (1), (2) i (3):

$$h^2 = \frac{6\,P\,l}{k\,b}\,, \tag{4}$$

a po wstawieniu do (4) danych liczbowych:

$$h^2 = \frac{6\,P \cdot 3,0}{30 \cdot 10^6 \cdot 0,12}\,, \tag{5}$$

czyli

$$h = 0,002236 \cdot \sqrt{P}\,. \tag{6}$$

Na postawie zależności (6) można sporządzić wykres $h(P)$ przedstawiony na rys.2.

Patronem honorowym OWT jest Minister Gospodarki.
Organizatorem OWT jest Federacja Stowarzyszeń Naukowo-Technicznych NOT.
Olimpiada jest finansowana ze środków MEN.

Wyniki umieszczono w tabeli poniżej:

P [kN]	h [m]
1	0,071
2	0,100
3	0,123
4	0,141
5	0,158
6	0,173

Spełnienie warunku c)

Ugięcie końca wspornika obciążonego siłą skupioną P wyraża wzór:

$$f = \frac{P\, l^3}{3\, E\, J}\,, \tag{7}$$

zaś moment bezwładności przekroju prostokątnego – wzór:

$$J = \frac{b\, h^3}{12}\,. \tag{8}$$

Po przekształceniach, otrzymuje się z (7) i (8) następującą zależność:

$$h^3 = \frac{P\, l^3\, 12}{3\, E\, b\, f} = \frac{4\, P\, l^3}{E\, b\, f}\,. \tag{9}$$

Po wstawieniu danych liczbowych do (9) jest:

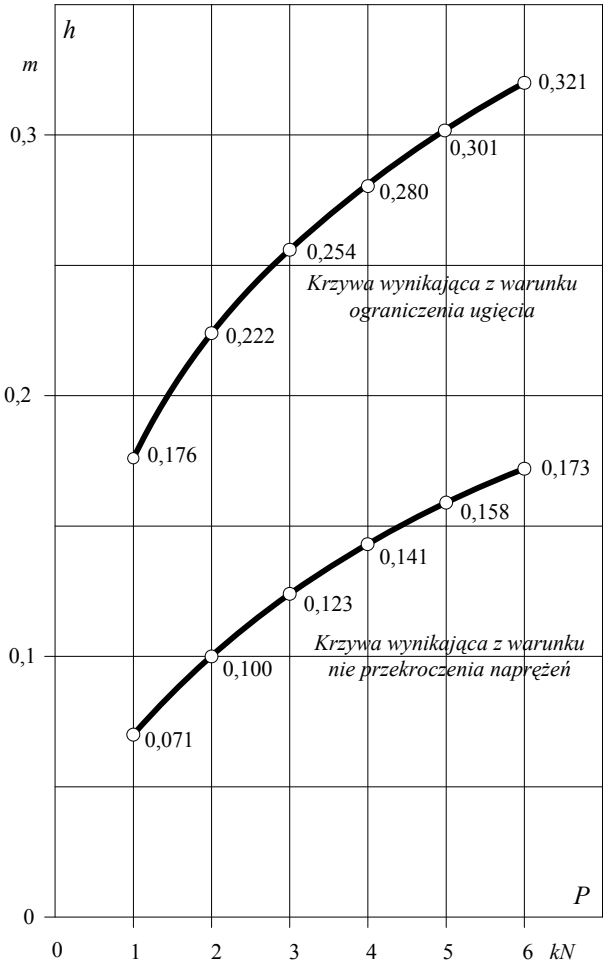
$$h^3 = \frac{4\, P \cdot 3^3}{11000 \cdot 10^6 \cdot 0,12 \cdot 0,015} = 0,0000545 \cdot P\,, \tag{10}$$

czyli

$$h = 0,01761 \cdot \sqrt[3]{P}\,. \tag{11}$$

Aby sporządzić wykres $h(P)$ (rys.2) należy wykonać na podstawie zależności (11) proste obliczenia, których wyniki są przedstawione w tabelce poniżej:

P [kN]	h [m]
1	0,176
2	0,222
3	0,254
4	0,280
5	0,301
6	0,321



Rys.2.

Z rysunku wynika wyraźnie, że decydujące znaczenie przy wyborze wysokości drewnianej belki ma warunek ograniczenia ugięcia końca wspornika.

Rozwiązanie zadania 2

Strumień ciepła oddawany przez piec

$$Q_p = F_p\, h_w \left(T_p - T_w\right) .$$

Strumień ciepła przez ścianę (od pomieszczenia do otoczenia)

$$Q_s = \frac{F_s}{R_s} \left(T_w - T_z\right) ,$$

gdzie

$$R_s = \frac{1}{h_w} + \frac{g_c}{\lambda_c} + \frac{g_{st}}{\lambda_{st}} + \frac{1}{h_z} . \tag{1}$$

Strumień ciepła przez okna

$$Q_{ok} = F_{ok}\, u_{ok} \left(T_w - T_z\right) ,$$

w stanie ustalonym

$$Q_p = Q_s + Q_{ok} ,$$

$$F_p\, h_w \left(T_p - T_w\right) = \frac{F_s}{R_s} \left(T_w - T_z\right) + F_{ok}\, u_{ok} \left(T_w - T_z\right) ,$$

$$T_p = \frac{\left(\dfrac{F_s}{R_s} + F_{ok}\, u_{ok}\right) \left(T_w - T_z\right)}{F_p\, h_w} + T_w , \tag{2}$$

$$R_s = \frac{1}{8} + \frac{0,37}{0,77} + \frac{0,05}{0,042} + \frac{1}{25} = 1,836 \left(\mathrm{m^2K}\right)/\mathrm{W},$$

$$T_p = \frac{\left(\dfrac{30}{1,836} + 6 \cdot 1,2\right) \cdot (20 - (-20))}{4,5 \cdot 8} + 20 = \frac{23,54 \cdot 40}{36} + 20 = 46^{\circ}\mathrm{C}$$

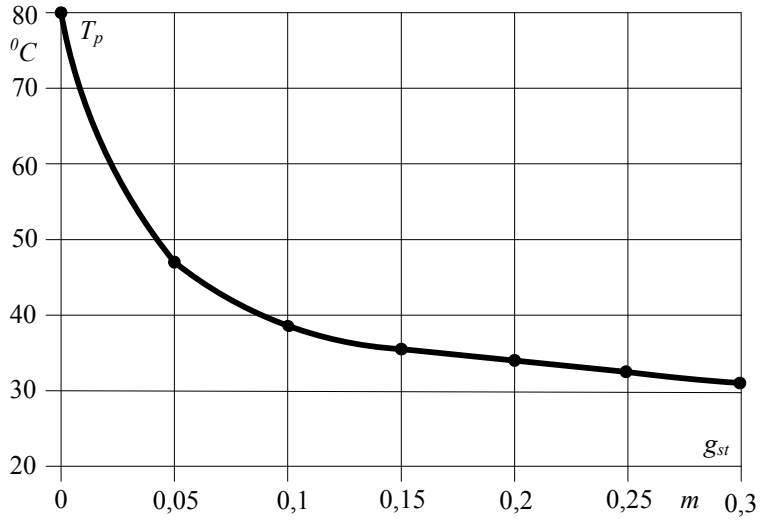
Celem otrzymania wykresu T_p w funkcji g_{st} najwygodniej wyznaczyć opór ściany dla kolejnych grubości styropianu (np.:0,01; 0,05; 0,10... 0,30) korzystając z wyrażenia (1):

$$R_s = 0,6455 + \frac{g_{st}}{0,042} ,$$

a następnie temperaturę wyliczać z (2):

$$T_p = \frac{33,33}{R_s} + 28\ .$$

g_{st} m	R $\left(\mathrm{m^2K}\right)/\mathrm{W}$	T_p $^\circ$C
0,00	0,646	80
0,05	1,836	46
0,10	3,026	39
0,15	4,217	36
0,20	5,407	34
0,25	6,598	33
0,30	7,788	32



Z wykresu wynika, że wpływ zwiększenia grubości styropianu na własności izolacyjne ściany silnie maleje przy grubościach powyżej 0,2 m.

Rozwiązanie zadania 3

Moc silnika:

Siła naciągu liny w układzie wielokrążka z jednym krążkiem przesuwnym:

$$F = \frac{m\,g}{2} = \frac{2000 \cdot 9,81}{2} = 9810 \text{ N} .$$

Moc mechaniczna silnika:

$$P = F\,v = 9810 \cdot 2 = 19620 \text{ W} = 19,62 \text{ kW}.$$

Sprawność silnika:

Temperatura końcowa przemiany adiabatycznego rozprężania („*suw pracy*"):

$$T_k = T_p \left(\frac{p_k}{p_p}\right)^{\frac{\kappa - 1}{\kappa}} = 1073 \cdot \left(\frac{1}{20}\right)^{\frac{1,4-1}{1,4}} = 456 \text{ K} ,$$

(uwaga: temperatury w kelwinach)

Sprawność obiegu Carnota $\left(T_G = 1073, \quad T_D = T_k = 456 \text{ K}\right)$:

$$\eta_C = 1 - \frac{T_D}{T_G} = 1 - \frac{456}{1073} = 0,575 .$$

Sprawność silnika

$$\eta_s = 0,5\,\eta_C = 0,5 \cdot 0,575 = 0,288 .$$

Strumień ciepła doprowadzonego w paliwie:

$$Q = \frac{P}{\eta_s} = \frac{19,62}{0,288} = 68,1 \text{ kW}.$$

Jednostkowe zużycie paliwa:

$$m_p = \frac{Q}{W} = \frac{68,1 \cdot 10^3}{42 \cdot 10^6} = 1,62 \cdot 10^{-3} \text{ kg/s} ,$$

$$m_p = 5,84 \text{ kg/h} .$$

Odp.: Moc mechaniczna silnika jest równa $19,62$kW przy jednostkowym zużyciu paliwa $5,84$kg/h.

Rozwiązanie zadania z optymalizacji

Oznaczenia:

x – liczba wytworzonych jednostek produktu O_1

y – liczba wytworzonych jednostek produktu O_2

x i y liczby całkowite, dodatnie.

Funkcja celu – zysk zakładu Z:
oznaczając przez k cenę jednostki produktu O_2

$$Z = 2 \cdot k \cdot x + k \cdot y .$$

Ograniczenia związane z wielkością zapasów magazynowych:
dla S_1:

$$12 \cdot x + 5 \cdot y \leq 60 \qquad \frac{x}{5} + \frac{y}{12} \leq 1 . \tag{1}$$

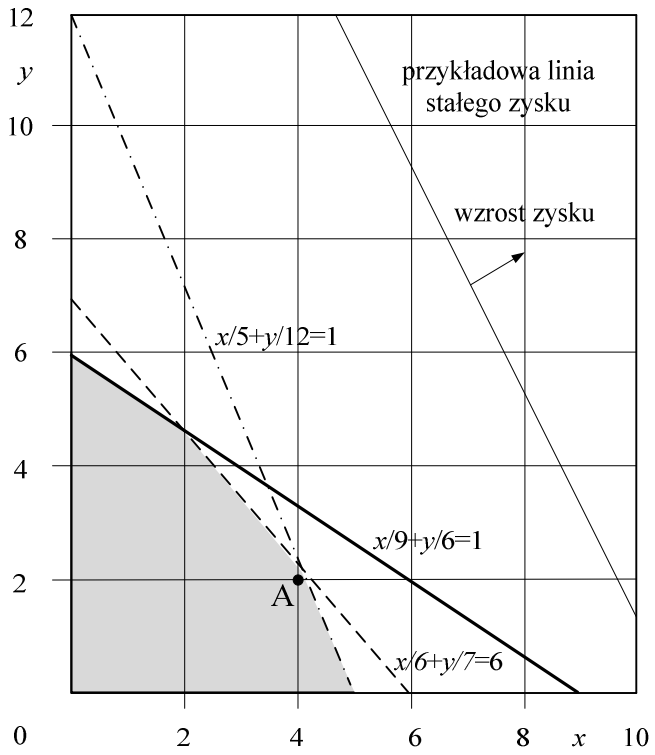
dla S_2:

$$7 \cdot x + 6 \cdot y \leq 42 \qquad \frac{x}{6} + \frac{y}{7} \leq 1 . \tag{2}$$

dla S_3:

$$6 \cdot x + 9 \cdot y \leq 54 \qquad \frac{x}{9} + \frac{y}{6} \leq 1 . \tag{3}$$

Rozwiązania nierówności (1) $\div$ (3) poszukuje się wykorzystując metodę wykreślną.



Obszar dopuszczalnych rozwiązań oznaczono na rysunku zacienionym polem. Naniesiono również na wykresie przykładową linię odpowiadającą stałemu zyskowi. Przesuwając tę linię w kierunku zacienionego pola widać, że pierwszym punktem o całkowitych wartościach współrzędnych w obszarze zacienionym jest punkt A. Odpowiada on największemu zyskowi możliwemu do uzyskania i stąd jest poszukiwanym rozwiązaniem. Współrzędne punktu A: $x = 4$, $y = 2$.

Odp: Maksymalny zysk zapewnia wyprodukowanie 4 jednostek produktu O_1 i 2 jednostek produktu O_2.

Rozwiązanie zadania z zastosowania informatyki

Dla siatki przedstawionej na rys.1. w treści zadania tablica wyjściowa będzie miała postać:

1	1	0	1	1	0	0	0	0
1	1	1	0	1	1	0	0	0
0	1	1	0	0	1	0	0	0
1	0	0	1	1	0	1	1	0
1	1	0	1	1	1	0	1	1
0	1	1	0	1	1	0	0	1
0	0	0	1	0	0	1	1	0
0	0	0	1	1	0	1	1	1
0	0	0	0	1	1	0	1	1

Opis algorytmu:

Działanie programu można przedstawić jako następującą sekwencję czynności:

1. Wczytaj siatkę (wystarczy zapamiętać tylko numery węzłów w elementach lub nawet wykonywać zawartość pętli 3 w trakcie czytania)
2. Zainicjuj pustą listę przechowującą niezerowe elementy tablicy
3. Zapamiętaj elementy niezerowe:
   - Iteruj po elementach
     - Iteruj po węzłach elementu (i)
       - Iteruj po węzłach elementu (j)
         - Zapisz niezerowy element (i, j)
4. Posortuj listę niezerowych elementów tablicy
5. Wypisz wzór wypełnienia tablicy:
   - Iteruj po wierszach tablicy (i)
     - Iteruj po kolumnach (j)
       - jeśli element (i, j) znajduje się na liście wypisz 1
       - jeśli go nie ma – wypisz 0
     - Wypisz znak nowego wiersza

Największą trudnością jest odpowiednie przechowanie listy niezerowych elementów.
Według efektywności możliwe rozwiązania można uszeregować następująco:

1. Hasz (tablica mieszająca) – rozwiązanie najlepsze
2. Drzewo poszukiwań binarnych (BST) lub dynamiczna tablica sortowana szybkim algorytmem sortowania – rozwiązanie średnie
3. Lista liniowa – rozwiązanie dostateczne

Przykładowy kod:
(Język C, bez wykorzystywania żadnych bibliotek poza standardową, przechowywanie elementów niezerowych w prostym haszu, niemal bez diagnostyki błędów)

```
#include <stdio.h>
#include <stdlib.h>
#define MIN(i,j) ((i)<(j)?(i):(j))
/* mniejsza z dwóch liczb */
#define MAX(i,j) ((i)>(j)?(i):(j))
/* większa z dwóch liczb */
/* dynamiczny wektor przechowujący niezerowe elementy jednego wiersza */
typedef struct {
   int v;
/* elementy */
   size_t s;
/* wielkość v */
   size_t n;
/* wypełnienie v */
} darr_t;

void resize_darr( darr_t *da ) {
/* powiększanie (dwukrotne) dynamicznego wektora */
   int *nv = realloc( da->v, 2*da->s*sizeof *nv );
   if( nv == NULL ) {
      fprintf( stderr, "Error in resize_darr\n" );
      exit( EXIT_FAILURE );
   }
   da->s *= 2;
}

int exist( darr_t *hash, int _i, int _j ) {
/* czy hasz przechowuje niezerowy element (i,j) ?*/
/* zwraca 1 jeśli tak, 0 jeśli nie */
   int i = MIN(_i,_j);
   int j = MAX(_i,_j);
   int k;
   for( k= 0; k < hash[i].n; k++ )
      if( hash[i].v[k] == j )
         return 1;
   return 0;
}
```

```
void add_non_zero( darr_t *hash, int _i, int _j ) {
/* dodanie elementu (i,j) do hasza */
   int i = MIN(_i,_j);
   int j = MAX(_i,_j);
   if( ! exist( hash,  i, j ) ) {
      if( hash[i].n == hash[i].s )
         resize_darr( hash+i );
      hash[i].v[hash[i].n++]= j;
   }
}

int main( int argc, char **argv ) {
   int n1, n2, n3;
   int nn;
   int ne;
   double x,y;
   FILE *in = argc > 1 ? fopen( argv[1], "r" ) : stdin;
   int i,j;
   darr_t *hash;
   fscanf( in, "%d", &nn );
/* czytamy liczbę węzłów */
   hash = malloc( nn * sizeof *hash );
   for( i= 0; i < nn; i++ ) {
      hash[i].v = malloc( 8*sizeof *hash[i].v );
      hash[i].s = 8;
      hash[i].n = 0;
   }
   for( i= 0; i < nn; i++ ) {
/* pomijamy współrzędne */
      fscanf( in, "%lf %lf", &x, &y );
   }
   fscanf( in, "%d", &ne );
/* czytamy liczbę elementów */
   for( i= 0; i < ne; i++ ) {
/* czytamy nr-y węzłów w elementach
   i od razu zapamiętujemy niezerowe elementy tablicy */
      fscanf( in, "%d %d %d", &n1, &n2, &n3 );
      add_non_zero( hash, n1, n2 );
      add_non_zero( hash, n1, n3 );
      add_non_zero( hash, n2, n3 );
   }
   fclose( in );
/* zamykamy plik wejściowy */
```

```
   for( i= 1; i <= nn; i++ ) {
/* pętla po wierszach tablicy */
      for( j= 1; j < i; j++ )
/* pętla po kolumnach przed diagonalą */
         if( exist( hash, i, j ) )
            printf( " 1" );
         else
            printf( " 0" );
      printf( " 1" );
/* element diagonalny jest zawsze niezerowy */
      for( j= i+1; j <= nn; j++ )
/* pętla po kolumnach za diagonalą */
         if( exist( hash, i, j ) )
            printf( " 1" );
         else
            printf( " 0" );
         printf( "\n" );
   }
   for( i= 0; i < nn; i++ )
/* zwolnienie pamięci */
      free( hash[i].v );
   free( hash );
   return EXIT_SUCCESS;
}
```